UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Gustavo Fortes Tondello

ESPECIFICAÇÃO SEMÂNTICA DE QoS: A ONTOLOGIA QoS-MO

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação.

Prof. Frank Augusto Siqueira, Dr.

Orientador

Florianópolis, agosto de 2008

ESPECIFICAÇÃO SEMÂNTICA DE QoS: A ONTOLOGIA QoS-MO

Gustavo Fortes Tondello

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Frank Augusto Siqueira, Dr. (coordenador do PPGCC)

Banca Examinadora

_____

Prof. Frank Augusto Siqueira, Dr. (orientador)

_____

Prof. Carlos Barros Montez, Dr.

_____

Prof. Renato Fileto, Dr.

_____

Prof. Roberto Willrich, Dr.

"*Há um estímulo grandioso que move a vida humana.*
Esse estímulo é seu fim, é sua meta, é o todo;
esse estímulo é o que a incita continuamente
à busca do saber, do conhecimento."

(Carlos Bernardo González Pecotche – RAUMSOL)

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho apresenta a ontologia QoS-MO que permite a especificação de características e requisitos de QoS para Web Services Semânticos ou Componentes de Software e pode ser facilmente utilizada para estender a OWL-S ou outras ontologias de descrição funcional de Componentes.

As especificações de QoS modeladas a partir da ontologia QoS-MO podem ser utilizadas no projeto, desenvolvimento, publicação e descoberta de Web Services ou Componentes de Software.

Um mecanismo de busca semântica de Web Services ou Componentes de Software foi especificado, voltado para a descoberta de Componentes que atendam a um conjunto definido de restrições de QoS, utilizando a ontologia QoS-MO e a linguagem de consulta SPARQL. Um protótipo deste mecanismo foi desenvolvido, contando tanto com uma interface de programação como uma interface Web.

Os testes realizados demonstraram que o mecanismo proposto é viável e apresenta um desempenho aceitável e que a ontologia definida é capaz de expressar características de QoS complexas.

A comparação com mecanismos de descoberta de Web Services de propostas similares demonstrou que a abordagem QoS-MO é mais simples e eficiente, pois não depende de nenhum algoritmo complexo para sua execução, apenas de um mecanismo de inferência simples e da linguagem SPARQL.

ABSTRACT

This work presents the QoS-MO ontology for the specification of QoS attributes and requisites of Semantic Web Services or Software Components. The QoS-MO ontology can be easily used to extend OWL-S or any other Software Components functional description ontology.

The QoS descriptions modeled with QoS-MO can be used in design, development, publication and discovery of Web Services or Software Components.

A semantic search mechanism for Web Services or Software Components has been specified with the goal to find Components that fulfill a set of defined QoS requirements. The QoS-MO ontology was used to build this mechanism, along with the SPARQL query language. A prototype of this mechanism has been created, with both a programming interface and a Web-based interface.

Tests were conducted and they have demonstrated that the proposed mechanism is viable and has an acceptable performance. They have also shown that the proposed ontology is capable of expressing complex QoS attributes.

The comparison of the QoS-MO approach with similar ones has shown that QoS-MO is simpler and more efficient, since it does not depend on any complex algorithm to find Web Services that match the QoS requirements: it only needs a simple inference engine and the SPARQL language.

1. INTRODUÇÃO

A Web Semântica é definida por BERNERS-LEE et al. (2001) como “uma extensão da Web atual, na qual é dado um significado bem definido para a informação, possibilitando que os computadores e as pessoas trabalhem em cooperação”.

Um Web Service é um componente de software projetado especificamente para suportar a interação entre sistemas que executam em diferentes máquinas em uma rede, que pode ser desde uma Rede Local até a própria Internet (W3C, 2004c).

A união das características dinâmicas dos Web Services com as tecnologias de especificação semântica da Web Semântica resulta nos Web Services Semânticos, que estendem a especificação dos Web Services com um conjunto preciso de informações semânticas. Devido à especificação semântica, a criação de mecanismos automáticos de publicação, descoberta, composição e execução de Web Services Semânticos fica mais fácil.

Existem diversas linguagens e padrões definidos para a especificação de Web Services Semânticos, dentre os quais se destacam as ontologias OWL-S (OWL-S, 2004) e WSMO (ESSI WSMO, 2006).

A especificação de um Web Service Semântico deve conter, além da descrição da funcionalidade que o Serviço fornece, a indicação da qualidade do serviço oferecido. A especificação de Qualidade de Serviço (QoS) de um Web Service indica quais são as características não funcionais do Serviço, expressas geralmente através de medidas quantificáveis, e que permitem que um cliente encontre um Serviço que, além de realizar a função desejada, seja capaz de fazer isto garantindo um certo nível de qualidade.

Já existem na literatura várias propostas de padrões para a especificação de Qualidade de Serviço de Web Services Semânticos. No entanto, cada uma das propostas apresentadas resolve uma parte do problema, não havendo sido criado ainda um padrão que seja capaz de expressar os diversos aspectos da Qualidade de Serviço dos Web Services de forma completa.

O presente trabalho apresenta a ontologia QoS-MO (*Quality of Service Modeling Ontology* – Ontologia de Modelagem de Qualidade de Serviço), criada para permitir a especificação completa de Qualidade de Serviço de Web Services Semânticos. Os diversos modelos de especificação de QoS existentes foram estudados com o propósito de trazer as melhores características de cada um para o cenário da Web Semântica e sanar as limitações existentes em propostas similares.

1.1. Objetivos do trabalho

O objetivo deste trabalho é o desenvolvimento da ontologia QoS-MO para a especificação de Qualidade de Serviço de Web Services Semânticos. A ontologia proposta será capaz de especificar todos os aspectos envolvidos na Qualidade de Serviço de um Web Service Semântico e permitirá que a descoberta automática de Serviços considere os requisitos de Qualidade de Serviço além dos requisitos funcionais desejados.

A necessidade para a criação da ontologia QoS-MO surge da observação de que os padrões existentes para definição de QoS de Web Services Semânticos em geral possuem fraquezas que impossibilitam a especificação completa de QoS com todas as questões envolvidas. Podemos citar como exemplos a possibilidade de especificar mapeamentos entre características de QoS semelhantes, porém especificadas de forma diferente, ou a incorporação de uma especificação de QoS como parte de outra, entre outras características que serão apresentadas ao longo do trabalho.

Após a especificação das características de QoS, é necessário fornecer também um padrão que possibilite a descoberta de Serviços que atendam a requisitos de QoS estabelecidos. Para cumprir com este objetivo, procuramos reutilizar ao máximo os padrões já existentes para a criação de um mecanismo de busca. Por isto, este trabalho irá abordar esta questão através da especificação de um mecanismo de busca capaz de converter os requisitos de QoS em uma consulta na linguagem SPARQL – padrão da W3C para realização de consultas sobre bases de conhecimentos representadas em RDF – e listar os Web Services que atendem aos requisitos a partir da execução desta consulta em uma Ontologia baseada em QoS-MO.

Foram desenvolvidas uma API (interface de programação) e uma ferramenta Web de busca de Web Services, para validar o uso da ontologia QoS-MO e do mecanismo de descoberta de Web Services utilizando consultas em linguagem SPARQL.

1.2. Metodologia

As seguintes atividades foram desenvolvidas ao longo do trabalho:

- Estudo da bibliografia e dos padrões existentes para especificação de QoS;
- Criação e definição da ontologia QoS-MO;
- Implementação de uma API de um mecanismo de busca de Web Services Semânticos a partir de suas características de QoS;
- Implementação de um protótipo de uma ferramenta Web para pesquisa baseada em QoS de Web Services Semânticos, utilizando a API desenvolvida;
- Comparação da expressividade das especificações de QoS feitas com a ontologia QoS-MO com a possibilidade de especificação de QoS em outras ontologias semelhantes;
- Análise de desempenho da API de busca.

1.3. Resultados Esperados

Espera-se que a ontologia QoS-MO possa contribuir para o estudo a respeito da definição de um padrão de especificação de QoS para Web Services Semânticos, auxiliando na abertura de espaço para a criação de uma proposta de padrão internacionalmente aceito para este fim, como hoje já ocorre com o OWL-S para a especificação funcional dos Web Services Semânticos.

1.4. Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma.

Os capítulos 2 e 3 apresentam uma análise do contexto no qual está inserido este trabalho. O capítulo 2 mostra o contexto de pesquisa atual na área de Web Services Semânticos. O capítulo 3 mostra o contexto de pesquisa na área de Qualidade de Serviço, assim como os trabalhos já realizados sobre a aplicação de modelos de QoS em Web Services Semânticos.

No capítulo 4 a ontologia QoS-MO é descrita e detalhada. Também são apresentados alguns exemplos de especificação de QoS utilizando a ontologia QoS-MO.

No capítulo 5 é descrita a API do mecanismo de busca sobre a ontologia QoS-MO e o protótipo de uma ferramenta Web de busca de Web Services Semânticos, construído

para verificar e validar as idéias apresentadas, assim como testar a viabilidade e o desempenho do mecanismo projetado.

O capítulo 6 analisa a contribuição científica proposta pela ontologia QoS-MO, comparando-a com outras ontologias já existentes para o mesmo fim, e apresentando os resultados dos testes de desempenho da API de busca.

No último capítulo são apresentadas as conclusões e as perspectivas de trabalhos futuros.

2. WEB SERVICES SEMÂNTICOS

Neste capítulo, iniciaremos a apresentação do contexto no qual se insere este trabalho. Inicialmente será apresentada uma visão geral da tecnologia de Web Services; em seguida, será apresentado o conceito de Web Semântica, seus objetivos e as tecnologias empregadas para sua construção; por fim, será descrita a tecnologia de Web Services Semânticos, na qual se baseia o presente trabalho.

2.1. Web Services

Na década de 90, a integração entre aplicações distribuídas era realizada utilizando-se padrões como CORBA (OMG, 1995), Java RMI (DWONING, 1998) e DCOM (BROWN, 1996). No entanto, estes padrões eram bastante limitados. Por exemplo, RMI era limitado a aplicações desenvolvidas em Java e DCOM era limitado a aplicações em plataformas da Microsoft. Estas limitações dificultavam muito a integração de aplicações desenvolvidas em plataformas diferentes, já que, dependendo do caso, uma aplicação em uma plataforma não conversava com uma de uma plataforma diferente. (PENNINGTON et al., 2007)

Para facilitar a integração de aplicações no ambiente da Web, foi desenvolvido na década de 2000 o conceito de Arquitetura Orientada a Serviços (SOA, do inglês *Service-Oriented Architecture*). Uma Arquitetura Orientada a Serviços é "um conjunto de componentes que podem ser invocados e cujas descrições de interface podem ser publicadas e descobertas. Os componentes são disponibilizados como serviços independentes que são acessados de forma padronizada." (PENNINGTON et al., 2007)

Uma Arquitetura Orientada a Serviços precisa ter os seguintes atributos (PENNINGTON et al., 2007):

- *Escalabilidade*: considerando a Web toda, a arquitetura pode ser utilizada dentro de uma organização, entre organizações ou envolvendo o mundo todo.
- *Baixo acoplamento*: as partes envolvidas na arquitetura podem enviar mensagens de forma independente do receptor. Isto evita que uma mudança em um serviço tenha grande impacto em outros.

- *Interoperabilidade*: as partes precisam se comunicar, independentemente da máquina ou plataforma na qual se encontram.
- *Descoberta*: um serviço deve ser capaz de escolher outro para se comunicar a partir de uma lista de serviços disponíveis, em um processo dinâmico.
- *Abstração*: a arquitetura deve ocultar detalhes tecnológicos, permitindo que os desenvolvedores se concentrem na lógica de negócio.
- *Padronização*: os protocolos de comunicação devem ser padronizados para permitir a máxima interoperabilidade.

A tecnologia de Web Services é uma das principais tecnologias empregadas para o desenvolvimento de uma arquitetura SOA. CURBERA et al. (2001) definem Web Service como "uma aplicação em rede que pode interagir utilizando protocolos Web aplicação-para-aplicação sobre interfaces bem definidas e que é descrita utilizando uma linguagem padrão de descrição funcional".

Um Web Service é um componente de software que pode ser invocado pela Web através de mensagens no formato XML (*eXtensible Markup Language* – linguagem de marcação extensível) (W3C, 2003a) seguindo, por exemplo, o padrão SOAP (*Simple Object Access Protocol* – protocolo simples de acesso a objetos) (W3C, 2003b).

Para que possa ser encontrado, um Web Service precisa ser descrito e publicado. A descrição é feita através do padrão WSDL (*Web Services Description Language* – linguagem de descrição de Web Services) (W3C, 2007b) e a publicação é feita em um registro chamado UDDI (*Universal Description Discovery & Integration* – descrição, descoberta e integração universal) (OASIS, 2004).

A tecnologia XML (*eXtensible Markup Language* – linguagem de marcação extensível) (W3C, 2003a) permite criar marcações personalizadas para as informações em um documento. A linguagem XML é usada como base para todas as linguagens utilizadas pelos Web Services, incluindo SOAP e WSDL.

Existem inúmeras formas de utilizar as informações em um documento XML, porém quem for utilizar a informação precisa saber o que cada marcação significa, pois a linguagem XML ocupa-se da estrutura do documento, não do seu significado.

2.1.1. Invocação: SOAP

O padrão SOAP (W3C, 2003b) define os formatos de mensagens XML que servem como protocolo de comunicação para a interação entre aplicações escritas em diferentes linguagens e que executam em diferentes plataformas.

A especificação do SOAP define um padrão de mensagem como a que segue.

```
<?xml version = ”1.0”?>
<soap:Envelope
  xmlns:soap =
    ”http://www.w3.org/2001/12/soap-envelope”
  soap:encodingStyle =
    ”http://www.w3.org/2001/12/soap-encoding”>
<soap:Header>
  ...
</soap:Header>
<soap:Body>
  ...
</soap:Body>
</soap:Envelope>
```

O envelope define o *namespace*[1] e o tipo de codificação utilizados; o cabeçalho (*Header*) é opcional e define informações adicionais sobre a mensagem; o corpo (*Body*) contém os dados da mensagem.

2.1.2. Descrição, Publicação e Descoberta: WSDL e UDDI

WSDL (W3C, 2007b) é a principal linguagem para a descrição de Web Services. Ela permite a definição da interface do serviço, para que ele possa ser invocado sem necessitar de conhecimentos sobre os detalhes de implementação do mesmo.

A descrição da interface de um Web Service inclui a assinatura de todas as operações oferecidas, definindo o nome da operação, as entradas, as saídas e as exceções. Além da interface, o documento WSDL descreve o serviço e as integrações possíveis.

---

[1] *Namespaces* XML são utilizados para garantir a unicidade de elementos e atributos nomeados em um documento XML. Eles são definidos pela recomendação W3C *Namespaces in XML*. (Fonte: http://en.wikipedia.org/wiki/XML_Namespace)

Segue um exemplo parcial de uma descrição em WSDL (Semantic Web Services Challenge, 2006 apud PENNINGTON et al., 2007):

```
<wsdl:definitions targetNamespace=”mooncompany”
xmlns:wsdl=”http://schemas.xmlsoap.org/wsdl/”
xmlns:wsdlsoap=”http://schemas.xmlsoap.org/wsdl/soap/”
xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
<wsdl:message name=”SearchCustomerResponseMessage”>
 <wsdl:part element=”impl:SearchCustomerResponse”
   name=”SearchCustomerResponse”/>
  </wsdl:message>
 <wsdl:portType name=”SearchCustomer”>
   <wsdl:operation name=”search”>
   <wsdl:input message=”impl:SearchCustomerRequestMessage”/>
   <wsdl:output
     message=”impl:SearchCustomerResponseMessage”/>
   </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name=”CRMServiceSoapBinding”
     type=”impl: SearchCustomer “>
   <wsdlsoap:binding style=”document”
     transport=”http://schemas.xmlsoap.org/soap/http”/>
   <wsdl:operation name=”search”>
     <wsdlsoap:operation soapAction=”search”/>
     </wsdl:operation>
   </wsdl:binding>
 <wsdl:service name=”CRMService”>
   <wsdl:port binding=”impl:CRMServiceSoapBinding”
     name=”CRMService”>
   <wsdlsoap:address
 location=”http://138.232.65.158/moon/services/CRMService”/>
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Este exemplo define um serviço cujo nome é `CRMService` e que é acessado pelo URL `http://138.232.65.158/moon/services/CRMService`. Ele possui uma

porta também nomeada `CRMService`, que é especificada pela ligação `CRMServiceSoapBinding`. Esta ligação, definida pelo tipo de porta `SearchCustomer`, possui uma operação chamada `search` e utiliza o protocolo de transporte de documento SOAP/HTTP. A operação `search` é definida com uma mensagem de entrada `SearchCustomerRequestMessage` e uma mensagem de saída `SearchCustomerResponseMessage`.

O UDDI (OASIS, 2004) define um Web Service padrão para a descoberta de descrições WSDL de outros Web Services. Um registro UDDI publica a descrição dos serviços para que um cliente possa localizar o serviço que irá invocar. O padrão UDDI é escalável, permitindo a descoberta eficiente de serviços relevantes em registros contendo milhares de Web Services.

2.2. Web Semântica

A quantidade de informações e serviços disponíveis na Web atual é cada vez maior. Os padrões utilizados para disponibilização destas informações e serviços foram todos criados com base em definições sintáticas, organizando as informações de forma a possibilitar que um computador as encontre. No entanto, o trabalho de interpretar e compreender, ou seja, extrair a semântica das informações, ainda necessita ser feito pelo ser humano.

A idéia por trás da Web Semântica, definida por BERNERS-LEE et al. (2001), é descrever e gerenciar toda a informação e os serviços disponíveis na Web de forma semântica, isto é, através de uma descrição completa da semântica da informação codificada em um formato que possa ser processado pelos computadores.

A evolução das tecnologias da Web Semântica abre campo para o desenvolvimento de um grande número de novas aplicações inteligentes.

DAVIES et al. (2006) apresentam alguns exemplos de aplicações que poderão ser construídas a partir da Web Semântica. O primeiro passo será organizar e encontrar informações baseadas no seu significado, ao invés de considerar apenas textos literais. Assim, por exemplo, uma pesquisa pelo termo “Jaguar”, dependendo do contexto semântico em que for feita, pode se concentrar apenas em resultados sobre veículos ou apenas sobre felinos.

Outra aplicação da Web Semântica apresentada por DAVIES et al. (2006) é a melhoria da forma de apresentação das informações. Por exemplo, as informações retornadas pela pesquisa por “Jaguar” poderiam ser agrupadas relacionando quais se referem a veículos, quais a felinos e quais a outros assuntos; a mescla das informações de todos os documentos relevantes com remoção de redundância e sumarização; e a apresentação de relações entre as principais entidades dos documentos encontrados de forma visual.

BERNERS-LEE et al. (2001) apresentam um exemplo de aplicação da Web Semântica ainda mais ousado. Em seu exemplo, agentes inteligentes conectados entre si através da Web Semântica são capazes de agendar, automaticamente, uma série de consultas para tratamento médico de uma pessoa, consultando as informações semânticas contidas no agente do médico – informações sobre o tratamento necessário –, nos agentes das clínicas – tratamentos disponíveis, horários – e nos agentes dos filhos da pessoa – agenda de horários disponíveis. Em poucos minutos, a tarefa estaria concluída pelos agentes inteligentes com uma quantidade mínima de intervenção humana necessária.

As tecnologias básicas para o desenvolvimento da Web Semântica, segundo BERNERS-LEE et al. (2001), são o XML e o RDF, apoiados por URI e Unicode. O documento original de 2001 apresentou uma visão dos padrões que formam parte da Web Semântica, porém BERNES-LEE apresentou em sua Palestra no AAAI2006 uma visão revisada destes padrões (BERNES-LEE, 2006), que é mostrada na Figura 1.

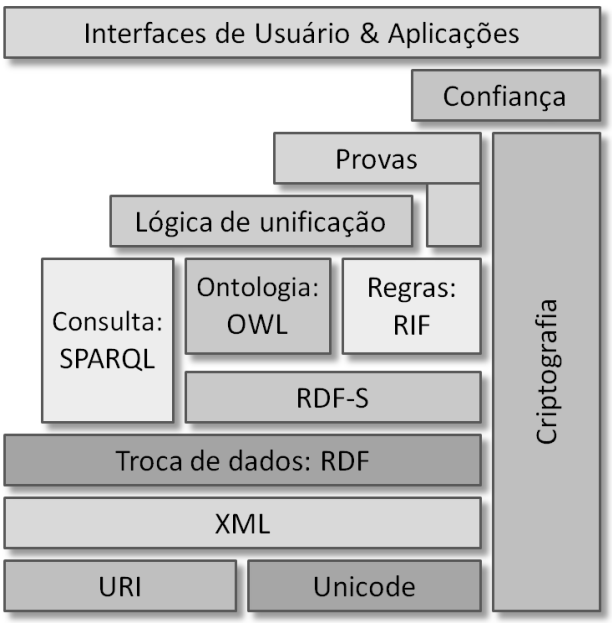

Figura 1: Arquitetura da Web Semântica (BERNERS-LEE, 2006)

2.2.1. Troca de dados: XML e RDF

A tecnologia RDF (*Resource Description Framework* – estrutura de descrição de recursos) (W3C, 2004a) é o padrão recomendado pela W3C para representar conhecimento, incluindo ontologias e anotações semânticas. A RDF codifica a informação em triplas que permitem a descrição de características ou propriedades de recursos identificados por URIs (*Uniform Resource Identifier* – identificador uniforme de recurso). Cada tripla descreve uma propriedade e é formada por sujeito, predicado e objeto (recurso, propriedade e valor da propriedade). BERNERS-LEE et al. (2001) mencionam que "esta estrutura é uma forma natural para descrever a vasta maioria dos dados processados pelas máquinas".

As triplas RDF podem ser representadas com diversas sintaxes, sendo uma das mais comuns a utilização de documentos XML.

O uso de URIs para a identificação dos recursos em RDF garante que cada recurso seja um conceito que pode ser encontrado por qualquer um na Web, não somente uma palavra que pode ter diversos significados, como geralmente ocorre quando a informação é expressa em linguagem natural.

2.2.2. Ontologias

Além de RDF, é necessária a existência de alguma forma de garantir que diferentes agentes utilizem o mesmo identificador para se referir a um conceito. A solução é a criação de ontologias.

O termo *ontologia* surgiu na Filosofia, onde se refere a teorias a respeito da natureza da existência das coisas, e uma disciplina que estuda estas teorias.

O termo foi adotado pelos pesquisadores de Inteligência Artificial e Web, passando a significar "um documento ou arquivo que formalmente define as relações entre termos" (BERNERS-LEE et al., 2001). Outra definição comumente aceita é "uma especificação formal e explícita de uma conceitualização em um domínio de interesse" (GRUBER, 1993), ou ainda, "uma descrição dos conceitos e relacionamentos que podem existir para um agente ou uma comunidade de agentes" (GRUBER, 1993).

Ontologias vêm sendo utilizadas na Computação com o propósito de permitir o compartilhamento e o reuso de conhecimento. Quando existe uma ontologia que descreve o conhecimento de um determinado domínio, agentes de software podem ser escritos

seguindo o que GRUBER (1993) chamou de um comprometimento ontológico, ou seja, um comprometimento em utilizar o vocabulário definido pela ontologia. Desta forma, diferentes agentes comprometidos com a mesma ontologia conseguem comunicar-se, já que seu vocabulário é entendido por todos, mesmo que os dados que cada agente tenha disponível não sejam os mesmos.

Uma ontologia é composta por conceitos ou classes, relações, instâncias e axiomas. Uma definição formal de uma ontologia é uma tupla (C, R, I, A) onde C é um conjunto de conceitos, R é um conjunto de relações, I é um conjunto de instâncias e A é um conjunto de axiomas (STAAB & STUDER, 2004 apud DAVIES et al., 2006).

Diversas linguagens foram criadas para a especificação de ontologias. A família de linguagens mais conhecida atualmente surgiu com as linguagens OIL (*Ontology Interchange Layer* – camada de troca de ontologias) (FENSEL et al., 2001) e DAML (*DARPA Agent Markup Language* – linguagem de marcação de agentes do DARPA) (DAML, 2003). Estas duas linguagens foram posteriormente combinadas, dando origem à linguagem DAML+OIL (CONNOLY et al., 2001) que, por sua vez, foi substituída pela linguagem OWL (*Web Ontology Language* – linguagem de ontologia da Web) (W3C, 2004b).

2.2.3. A linguagem de ontologias OWL

A OWL (W3C, 2004b) é uma linguagem para descrição de ontologias, projetada para ser compatível com a *World Wide Web* e com a Web Semântica em particular.

OWL é uma extensão de vocabulário de RDF/RDF-S para expressar de maneira padronizada construções e restrições mais sofisticadas em ontologias que as expressas somente com RDF/RDF-S. Esta linguagem oferece ainda as seguintes capacidades:

- habilidade de ser distribuída;
- escalabilidade para a Web;
- compatibilidade com padrões de acessibilidade e internacionalização;
- aberta e extensível.

OWL permite a definição de Classes e Propriedades, assim como o relacionamento entre classes (por exemplo, disjunção, união e intersecção), cardinalidades, relações de igualdade, classes enumeradas e tipos e características das propriedades.

A definição de uma classe em OWL corresponde à definição dos requisitos para que um indivíduo seja uma instância desta classe. Estes requisitos são descritos em termos das propriedades da classe e de restrições sobre a classe e suas propriedades. OWL define uma classe básica, chamada *Thing*, da qual derivam todas as demais classes.

As propriedades definem as características e os relacionamentos entre os indivíduos. Toda propriedade tem um domínio (*domain*), que indica quais classes têm a propriedade em sua definição, e um alcance (*range*), que indica que tipos de valores a propriedade pode assumir. Existem dois tipos de propriedade: as propriedades de tipos de dados, que têm como alcance um determinado tipo de dados da linguagem (os tipos básicos da OWL são *boolean*, *float*, *int*, *string*, *date*, *dateTime* e *time*); e as propriedades de objetos, que têm como alcance classes e instâncias de classes definidas em OWL.

Existem três sublinguagens de OWL, que crescem em expressividade, podendo ser utilizadas segundo as necessidades da aplicação: OWL *Lite*, OWL DL e OWL *Full*.

- *OWL Lite* é a mais simples, suportando apenas hierarquias de classificação e restrições simples. Ela pode ser considerada um subconjunto da OWL DL.
- *OWL DL* suporta o máximo de expressividade possível, mantendo a completude computacional e a decidibilidade. Esta sublinguagem contém todas as construções de OWL, porém existem restrições para o seu uso. A OWL DL é particularmente interessante, pois sua expressividade corresponde à das lógicas de descrição (BAADER et al., 2003), que formam a base formal da OWL.
- *OWL Full* permite a expressividade máxima, sem nenhuma restrição sintática, porém não há nenhuma garantia computacional. É improvável que possa ser desenvolvido um software que suporte inferência para todos os recursos de OWL *Full*.

2.2.4. Lógicas de Descrição e Inferência

As lógicas de descrição (BAADER et al., 2003) são linguagens para representação de conhecimento. MCGUINESS et al. (2002) definem que linguagens para representação de conhecimento são especificadas quando tanto sua sintaxe quanto sua semântica são descritas: a sintaxe descreve as construções válidas e a semântica descreve o significado de cada construção.

A linguagem OWL DL suporta as abordagens de lógica baseada em quadros (*frames*) e baseada em objetos, por isso as primitivas de modelagem são classes (ou quadros), propriedades e indivíduos. A existência de um indivíduo em uma descrição é uma afirmação da existência de algo no mundo real. As propriedades de um indivíduo especificam suas características. As classes são utilizadas para agrupar os indivíduos com características semelhantes, por exemplo, todos os indivíduos que são pessoas ou todos os indivíduos que têm cor vermelha. A definição de uma classe é a especificação de quais condições um indivíduo deve satisfazer para ser considerado membro da classe.

Inferência é o processo de descobrir novos conhecimentos a partir de conhecimentos existentes, com o objetivo de formar um modelo conciso do mundo real (BRUSEE & POKRAEV, 2007). A utilização de mecanismos de inferência permite obter um modelo completo a partir de uma ontologia, sem a necessidade de especificar cada uma das triplas para representar todo o conhecimento disponível, já que a especificação de apenas algumas triplas permite que as demais sejam inferidas.

As definições de classes, propriedades e indivíduos das lógicas de descrição permitem realizar inferência em cada um destes três níveis. A diferença entre estes níveis de inferência é a seguinte (BRUSEE & POKRAEV, 2007):

- *Inferência em nível de propriedades* implica inferir novas triplas a partir das existentes. Por exemplo, a existência de propriedades transitivas permite criar um grafo de triplas a partir das propriedades transitivas especificadas.
- *Inferência em nível de classes* implica em descobrir se uma classe *B* é uma subclasse de *A*. Isto é feito avaliando-se se as condições para ser um membro de *B* contém as condições para ser um membro de *A*. Também é possível identificar duas classes equivalentes, se os conjuntos para ser um membro de cada uma delas forem equivalentes. Utilizando este tipo de inferência, é possível construir a hierarquia de classes completa de uma ontologia.
- *Inferência em nível de indivíduos* implica em descobrir se um indivíduo pode existir em um modelo (verificação de consistência) e também de quais classes um indivíduo é membro (realização). Fazendo isto para apenas uma classe e descobrindo todos os indivíduos membros da classe, pode-se fazer a recuperação de indivíduos da classe, permitindo que o modelo lógico seja utilizado co-

mo um banco de dados. Isto pode ser feito através de linguagens de consulta, como a SPARQL.

2.2.5. A linguagem de consulta SPARQL

A linguagem SPARQL (W3C, 2007a) é uma linguagem que permite executar consultas sobre conjuntos de dados expressos em RDF. Ela tem a capacidade de especificar consultas na forma de grafos com componentes obrigatórios e opcionais, assim como conjunções e disjunções entre os componentes. O resultado de uma consulta SPARQL pode ser um conjunto de resultados ou um grafo RDF.

Geralmente, uma consulta SPARQL é baseada em um conjunto de padrões de triplas. Um padrão de tripla é semelhante a uma tripla em RDF, porém o sujeito, o predicado ou o objeto podem ser variáveis. A consulta é composta por duas partes: uma cláusula `SELECT`, que define quais são as variáveis que irão aparecer nos resultados, e uma cláusula `WHERE`, que define o padrão básico de grafo, com o conjunto de triplas padrão. O mecanismo de execução de SPARQL irá encontrar o subconjunto de triplas no RDF que satisfaçam à condição expressada pelo padrão básico de grafo, que pode conter triplas obrigatórias, triplas opcionais e restrição quanto aos valores literais das triplas.

Um exemplo de uma consulta SPARQL simples, de (W3C, 2007a):

Dados:

```
<http://example.org/book/book1>
<http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .
```

Consulta:

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1>
  <http://purl.org/dc/elements/1.1/title>
  ?title .
}
```

Resultado: um conjunto de resultados, contendo, no caso, somente o valor "`SPARQL Tutorial`", o título do recurso `<http://example.org/book/book1>`.

2.2.6. Troca de regras: RIF

A linguagem OWL já tem se estabelecido como um padrão para representação de conhecimento na Web Semântica. No entanto, se identificou a necessidade de estender a OWL com a possibilidade da especificação de regras lógicas mais amplas, para suportar uma variedade maior de aplicações (HITZLER et al., 2005). Uma das iniciativas para combinar ontologias OWL com regras lógicas é a linguagem SWRL – *Semantic Web Rule Language* (Linguagem de Regras da Web Semântica) (HORROCKS et al., 2004).

A partir de 2005, foi estabelecido pela W3C um grupo de trabalho para a criação de um padrão para especificação e troca de regras entre diferentes agentes, o RIF – *Rule Interchange Format* (Formato de Troca de Regras) (W3C, 2005).

2.3. Web Services Semânticos

Os Web Services, como peças de funcionalidade que provêem serviços acessíveis pela Web, criaram um novo nível de funcionalidade na Web em direção à completa integração de componentes de software distribuídos através de padrões Web (DAVIES et al., 2006).

No entanto, a tecnologia atual de Web Services, em especial os padrões WSDL e UDDI, apesar de suportar a interoperabilidade entre sistemas de diferentes plataformas e escritos em diferentes linguagens, opera apenas no nível sintático, não sendo capaz de eliminar a necessidade de interação humana para encontrar e selecionar os Web Services desejados e determinar como combiná-los corretamente.

A tecnologia de Web Services Semânticos combina a característica dinâmica que os Web Services acrescentaram à Web com a característica semântica que vem sendo desenvolvida pela Web Semântica. A descrição semântica de Web Services permite a automatização de tarefas como descoberta, composição e execução de Web Services (MCILRAITH et al., 2001). A Figura 2 ilustra isto (FENSEL et al., 2007).

Para fornecer um *framework* completo para Web Services Semânticos, diversas iniciativas têm surgido nos últimos anos. A seguir, apresentaremos duas das mais importantes abordagens existentes: a abordagem WSMO (WSMO, 2008) e a ontologia OWL-S (OWL-S, 2004).

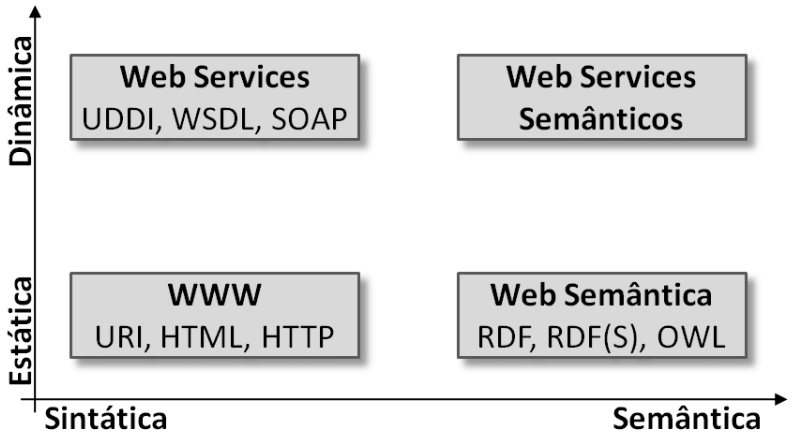

Figura 2: Evolução da Web (FENSEL et al., 2007)

2.3.1. A Abordagem WSMO

A WSMO (WSMO, 2008) é uma grande iniciativa para o estudo de modelos conceituais, representações formais e ambientes de execução para Web Services semânticos. Os três elementos que compõem a abordagem WSMO são: a ontologia WSMO (*Web Service Modeling Ontology* – Ontologia de Modelagem de Web Services) – um modelo conceitual para Web Services semânticos –, a linguagem WSML (*Web Service Modeling Language* – Linguagem de Modelagem de Web Services) – que provê uma sintaxe e semântica formal para a WSMO – e o ambiente WSMX (*Web Service Modeling Execution Environment* – Ambiente de Execução de Modelagem de Web Services).

A WSMO (ESSI WSMO, 2006) provê a especificação ontológica dos Web Services Semânticos. Os elementos da WSMO são definidos em uma linguagem de meta-meta-modelo baseada no MOF – *Meta Object Facility* (OMG, 2002).

A WSMO define os seguintes elementos:

- *Ontologias*: para a definição da semântica da terminologia utilizada pelos demais componentes da WSMO. As ontologias são especificadas por conceitos, relações, funções, instâncias e axiomas. Além disso, uma ontologia pode importar outras.
- *Web Services*: a definição dos Web Services é feita através de capacidades, ou seja, as funcionalidades oferecidas pelo serviço, e interfaces, isto é, como interagir com o serviço (coreografia) e como o serviço se comporta na composição com outros serviços para obter sua funcionalidade completa.

- *Objetivos*: especificam o que o cliente busca quando procura um serviço. Um objetivo é especificado por capacidades requeridas e interfaces requeridas.
- *Mediadores*: o serviço de mediação em WSMO é utilizado para resolver problemas de heterogeneidade entre elementos que devem operar em conjunto. Um mediador especifica os elementos de origem e destino e qual é o serviço que irá implementar a mediação.

As definições de Web Services, Objetivos e Mediadores podem importar ontologias que especificam a terminologia utilizada. Todos os elementos contêm propriedades não funcionais, baseadas no Conjunto de Metadados *Dublin Core* (DC) (WEIBEL et al., 1998), para a definição de informações genéricas.

A WSML (DE BRUIJN, 2005) é a linguagem para a descrição dos elementos do modelo conceitual WSMO. A linguagem WSML foi desenvolvida desde suas bases, não sendo baseada nos padrões atuais da Web Semântica (ROMAN et al., 2006).

A abordagem WSMO ainda inclui o WSMX (CIMPIAN et al., 2005), que é o ambiente de execução que permite a descoberta, seleção, mediação e invocação de Web Services Semânticos, além de ser a implementação de referência da WSMO).

2.3.2. A Ontologia OWL-S

OWL-S (OWL-S, 2004) é uma ontologia OWL de alto nível que permite a descrição das propriedades e capacidades de um Web Service de forma que possa ser processada por um computador. Desta forma, ela permite a automatização de tarefas como descoberta, execução, interoperabilidade, composição e monitoramento de Web Services Semânticos.

A ontologia OWL-S consiste em três sub-ontologias relacionadas: conhecidas como *Profile*, *Process Model* e *Grouding*.

Em OWL-S, um serviço é descrito por uma instância da classe `Service`, um conceito de alto nível que serve como ponto inicial para a definição de um Web Service. A peça chave da descrição de um Serviço é o seu perfil, descrito por uma instância da classe `ServiceProfile` (sub-ontologia *Profile*). A OWL-S não especifica como um serviço deve ser descrito, o que pode ser definido através da criação de subclasses de `ServiceProfile`. A descrição padrão, feita através da classe `Profile`, compreende a

descrição da organização que provê o serviço, a funcionalidade que o serviço executa e as características do serviço. As funcionalidades do serviço são descritas pelas entradas, saídas, pré-condições e efeitos de cada funcionalidade.

O perfil de um serviço OWL-S é descrito através de funcionalidades atômicas, enquanto o modelo do serviço (classe `ServiceModel`, da sub-ontologia *Process Model*) descreve o estado do serviço como um processo complexo de interação. O modelo pode ser descrito através de uma instância da classe `Process`, que especifica como um cliente pode interagir com o serviço. A descrição do processo é feita através de construções de *workflow*, como seqüências, condições, execuções paralelas, repetições, etc.

Finalmente, a especificação de um serviço OWL-S contém também o *grounding* (classe `ServiceGrounding`, da sub-ontologia *Grounding*), que é o mapeamento da especificação abstrata do serviço para sua especificação concreta, ou seja, a definição dos detalhes necessários para que o serviço possa ser invocado. O *grouding* permite mapear uma especificação em OWL-S para uma especificação em WSDL.

A Figura 3 mostra um exemplo dos indivíduos definidos em uma ontologia para a definição em OWL-S do Web Service *BravoAir* (OWL-S, 2004).

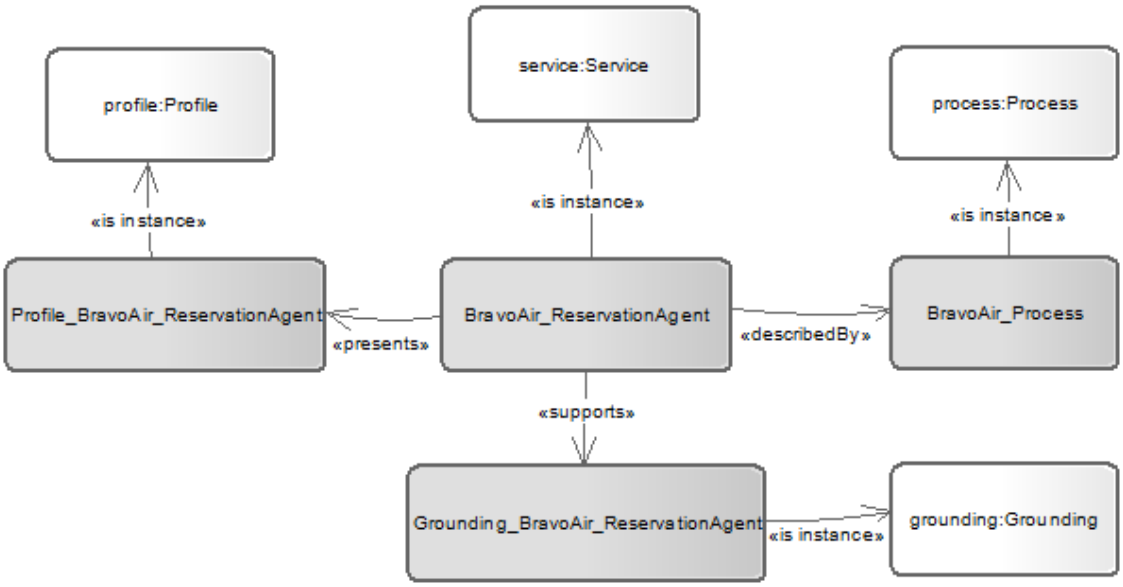

Figura 3: Exemplo de um Web Service definido em OWL-S

3. QUALIDADE DE SERVIÇO

Neste capítulo, seguimos apresentando o contexto no qual se insere este trabalho, através da definição de Qualidade de Serviço e os principais desafios da área. Em seguida, analisaremos a aplicação de especificações de Qualidade de Serviço no contexto de Web Services Semânticos e os trabalhos já realizados neste aspecto, estabelecendo as necessidades que ainda existem na área e que justificam o desenvolvimento do presente trabalho.

3.1. Definições

O termo Qualidade de Serviço (QoS, do inglês *Quality of Service*) é definido pela ISO como "um conjunto de qualidades relacionadas ao comportamento colaborativo de um ou mais objetos" (ISO, 1998). VOGEL et al. (1995) utilizam uma definição um pouco mais detalhada: "Qualidade de Serviço representa o conjunto de características quantitativas e qualitativas de um sistema multimídia distribuído necessárias para suportar as funcionalidades requeridas por uma aplicação".

No contexto de Web Services, requisitos de QoS são utilizados pela W3C (2003c) para se referir ao aspecto qualitativo de um Web Service, o que pode incluir desempenho, confiabilidade, escalabilidade, capacidade, robustez, tratamento de exceções, precisão, integridade, acessibilidade, disponibilidade, interoperabilidade, segurança e requisitos relacionados a rede. (W3C, 2003c) De forma semelhante, MENASCÉ (2002) define QoS como uma combinação de diversas qualidades ou propriedades de um serviço, como disponibilidade, segurança, tempo de resposta e *throughput* (capacidade de fluxo).

Para nosso trabalho, definiremos QoS como um conjunto de características ou propriedades de um determinado Componente de Software ou Web Service que são necessárias para que este consiga fornecer os seus serviços dentro dos parâmetros de qualidade aceitáveis para o seu cliente.

3.1.1. Especificação de QoS

A qualidade oferecida por um Componente ou Serviço é especificada através de um conjunto de *Características de QoS*, que são os aspectos quantificáveis de QoS,

como largura de banda, tempo de resposta, etc., e um conjunto de *Medidas de QoS*, que são os diferentes valores observados para cada uma das características.

A qualidade desejada por um Cliente precisa ser especificada como um conjunto de *Parâmetros de QoS* ou *Requisitos de QoS*. Os parâmetros podem ser especificados de várias formas diferentes, mas em geral eles têm a função de impor algumas restrições aos valores das medidas de cada uma das características de QoS.

Algumas linguagens existentes para especificação de QoS são: *QoS Description Language* – QDL (ZINKY et al., 1997), *Quality of Service Modeling Language* – QML (FRØLUND & KOISTINEN, 1998), *Web Service Level Agreement* – WSLA (LUDWIG et al., 2003) e *QoS Specification Language* – QSL (SIQUEIRA, 2002). Apesar de fornecerem mecanismos para a especificação de QoS, estas linguagens não possuem a flexibilidade fornecida por uma ontologia para o tratamento do significado semântico das restrições de QoS.

3.1.2. Mapeamento de QoS

Quando diferentes aplicações ou diferentes níveis de abstração enxergam qualidade através de diferentes características de QoS, é necessário que exista um mecanismo de mapeamento que permita converter os valores medidos de uma característica para outra, para que as estas façam sentido a todos os envolvidos (VOGEL et al., 1995).

Um exemplo de mapeamento necessário entre diferentes aplicações poderia ser quando uma aplicação especifica seu tempo de resposta em uma única característica que representa o tempo total decorrido entre o início e o fim da conversação, enquanto outra aplicação poderia considerar a existência de três diferentes tempos: o tempo que ela utiliza recebendo a requisição, o tempo gasto no processamento, e o tempo gasto para gerar e enviar a resposta.

Um exemplo de mapeamento entre níveis diferentes de abstração poderia ser a conversão de características de resolução de vídeo e profundidade de cor de uma aplicação em um valor de largura de banda necessária para a transmissão de dados (para serem entendidas pelo nível de rede).

3.1.3. Negociação e garantia de QoS

Quando dois Componentes ou Serviços vão interagir em uma composição, é necessário que seja feita uma comparação das características oferecidas e requisitos dese-

jados de QoS de cada elemento envolvido, procurando encontrar um conjunto de parâmetros que satisfaça a todos. No início da composição, pode ser definido um *Contrato de QoS* especificando os parâmetros acordados entre as partes e que devem ser respeitados ao longo do seu funcionamento.

A existência de um Contrato de QoS implica ainda na existência de algum mecanismo que permita realizar algum tipo de reserva e/ou monitoramento dos recursos computacionais (processador, memória, banda de rede, etc.) necessários para que o serviço possa ser executado dentro dos níveis de QoS estabelecidos (NAHRSTEDT & STEINMETZ, 1995).

Por isso, além da especificação de características e parâmetros de QoS, também é necessário que esteja disponível uma Arquitetura de QoS que atue como *middleware* capaz de realizar todo o procedimento de negociação de QoS e garantia de operação dos diversos componentes envolvidos dentro dos contratos definidos.

O processo de negociação e estabelecimento de um Contrato de QoS insere um *overhead* (tempo adicional) na utilização de um serviço. Este tempo é justificável no caso em que um cliente pretenda invocar mais de uma vez um determinado Web Service, de forma que o tempo gasto na primeira vez para selecionar o Web Service e estabelecer os parâmetros de operação resulte em um ganho de desempenho nas próximas vezes, que compense a perda inicial, ou ainda nos casos em que o parâmetro de qualidade que está sendo negociado – por exemplo, segurança ou confiabilidade – tenha uma importância que justifique o tempo gasto.

3.1.4. Acordos de Nível de Serviço

Um Acordo de Nível de Serviço (*Service Level Agreement* - SLA) é “uma definição formal da relação que existe entre um provedor de serviço e seu cliente” (VERMA, 2004). De maneira geral, um SLA descreve o que o cliente espera do fornecedor, as obrigações do cliente e do provedor, as restrições de desempenho, disponibilidade, segurança, etc. do serviço, além dos procedimentos necessários para assegurar a adequação aos parâmetros acordados no SLA.

Acordos de Nível de Serviço geralmente são utilizados quando uma empresa contrata fornecedores externos para realizar tarefas que estão fora de seu escopo de compe-

tências principais. Segundo VERMA (2004), os principais componentes de um SLA são:

- A descrição do serviço a ser fornecido;
- O nível de desempenho esperado do serviço, especificando sua confiabilidade e os tempos de resposta;
- Os procedimentos para reportar problemas no serviço;
- O tempo limite para que o fornecedor resolva o problema identificado;
- O processo para monitorar e reportar o nível de desempenho do serviço;
- As conseqüências para o fornecedor do serviço, caso ele deixe de atender aos requisitos acordados;
- Cláusulas de exceções e restrições de aplicabilidade do acordo.

SLAs também podem ser utilizados no contexto de Web Services. Neste caso, é necessário especificar os parâmetros de Qualidade de Serviço que serão garantidos durante a execução do serviço e também deverá existir um ambiente de execução que permita monitorar os níveis de qualidade e detectar possíveis falhas no cumprimento do acordo. Como se pode ver, este tipo de acordo engloba os Contratos de QoS, mencionados na seção anterior. Porém um SLA vai além de um simples Contrato de QoS, pois, além de especificar os níveis de QoS, o SLA especifica como monitorar e gerenciar o acordo e como penalizar as partes que deixarem de cumpri-lo.

Um exemplo de *framework* para definição de acordos de nível de serviço para Web Services é o *WSLA Framework* (KELLER & LUDWIG, 2003).

3.2. Metamodelo de QoS da OMG

O metamodelo do *framework* de QoS da OMG (2006) define uma linguagem abstrata para uma linguagem de modelagem que suporta a modelagem de conceitos de QoS. Ele foi desenvolvido como um metamodelo para a definição do *Profile* de QoS para UML. Este metamodelo é largamente aceito em sua área de pesquisa devido à sua adequação para a modelagem dos conceitos de QoS e é também adotado como a referência básica para a definição da ontologia QoS-MO.

O metamodelo de QoS da OMG define três pacotes. O pacote *QoS Characteristics* contém os elementos de modelagem para a descrição de características de QoS, di-

mensões de QoS (diferentes formas de quantificar uma característica) e contextos de QoS (restrições de qualidade que combinam diversas características de QoS). O pacote *QoS Constraints* inclui os elementos de modelagem para a descrição de contratos e restrições de QoS. O pacote *QoS Levels* inclui os elementos de modelagem para a especificação de níveis e transições de QoS. A seguir serão descritos cada um destes pacotes.

3.2.1. O pacote *QoS Characteristics*

Os conceitos envolvidos neste pacote são:

- *QoS Characteristic*: representam as características quantificáveis de um elemento de software, como latência, capacidade de fluxo (*throughput*), capacidade, escalabilidade, disponibilidade, confiabilidade, segurança, confidencialidade, integridade, probabilidade de erros, precisão e carga. Podem ser estendidas/especializadas através da relação `Sub-Parent`.
- *QoS Dimension*: são as dimensões para quantificação das características. Cada característica pode ter mais de uma forma de medi-la. Na especificação de uma dimensão, é importante definir sua ordenação (crescente ou decrescente), através da propriedade `direction`, sua unidade de medida, através da propriedade `Unit`, e sua qualificação estatística (por exemplo, máximo, mínimo, média, etc.), através da propriedade `statisticalQualifier`, para que o significado do valor da dimensão pode ser totalmente compreendido e comparado com outros.
- *QoS Category*: fornece uma forma de agrupar as características em categorias pré-definidas.
- *QoS Value*: utilizada quando o valor de uma característica pode ser definido em tempo de projeto, instancia uma *QoS Characteristic* e fixa os valores de suas definições (*QoS Dimension Slot*).
- *QoS Dimension Slot*: representa o valor primitivo de uma dimensão ou uma referência a outro valor.
- *QoS Context*: um contexto de QoS representa uma restrição de QoS composto por uma ou mais características de QoS. Uma característica de QoS é um contexto de QoS composto somente por ela própria, porém é possível definir contextos compostos por outros contextos ou características de QoS.

A Figura 4 mostra as metaclasses do modelo de características de QoS, a Figura 5 mostra as metaclasses do modelo de valores de QoS e a Figura 6 mostra as metaclasses do modelo de contextos de QoS (OMG, 2006).

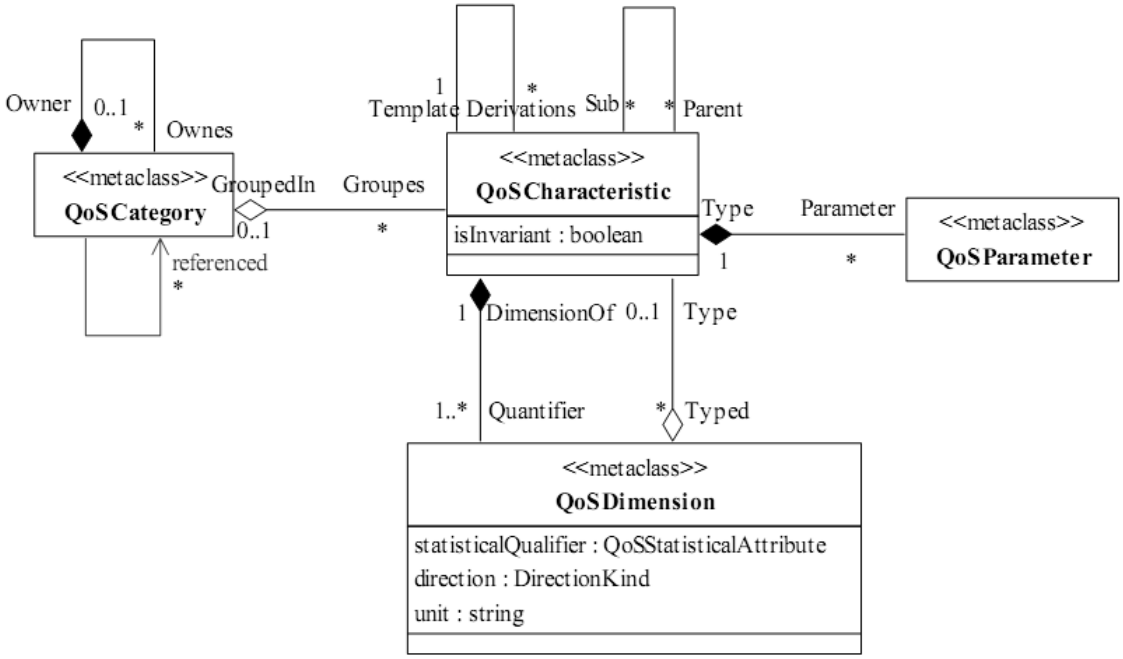

Figura 4: Modelo de características de QoS da OMG (2006)

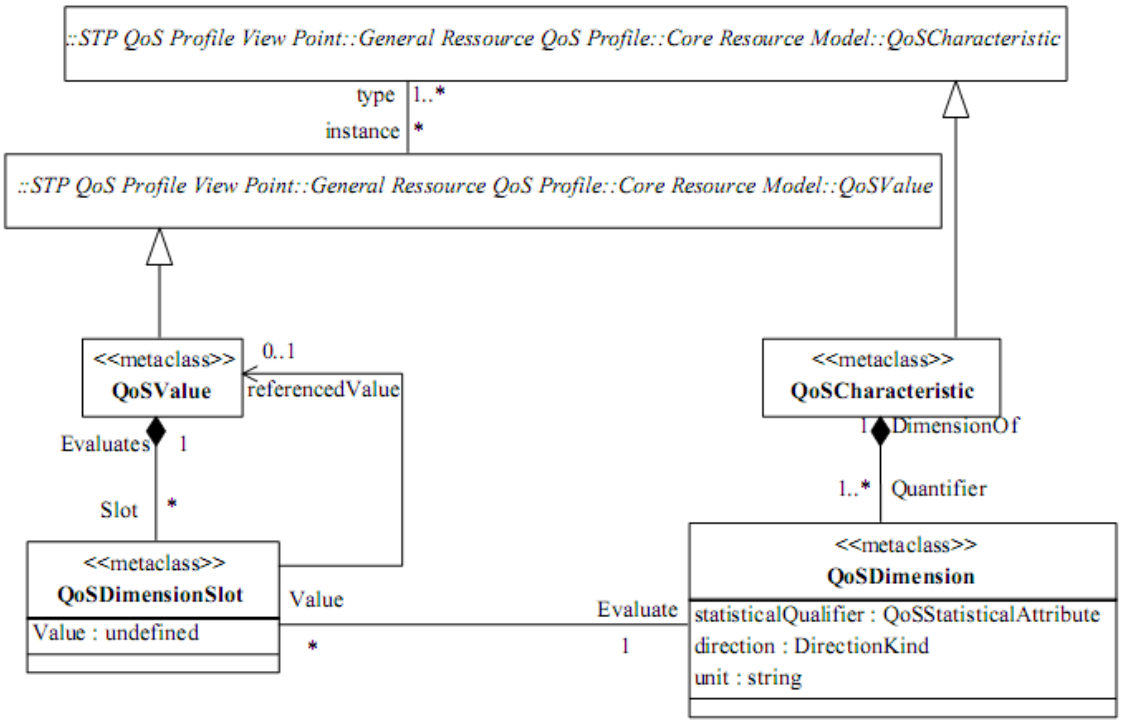

Figura 5: Modelo de valores de QoS da OMG (2006)

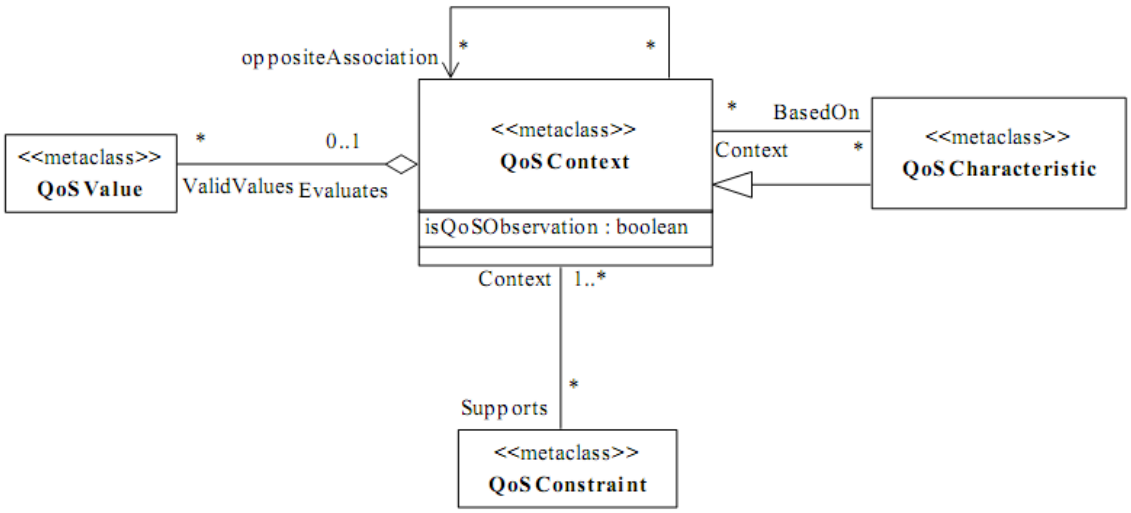

Figura 6: Modelo de contextos de QoS da OMG (2006)

3.2.2. O pacote *QoS Constraints*

Os conceitos envolvidos neste pacote são:

- *QoS Constraint*: metaclasse abstrata, utilizada para limitar os valores permitidos de uma ou mais *QoS Characteristics*. Esta metaclasse é utilizada em conjunto com *QoS Context*, que define as características envolvidas na restrição, enquanto *QoS Constraint* define os valores limite das restrições. Restrições de QoS podem ser do tipo *QoS Required*, *QoS Offered* ou *QoS Contract*.
- *QoS Required*: quando um cliente define uma restrição deste tipo, o fornecedor deve prover serviços que atendam aos requisitos do cliente; quanto é o fornecedor que define esta restrição, significa que o cliente deve atender a alguns requisitos para ter a qualidade oferecida, por exemplo, a freqüência máxima de invocação do serviço por parte do cliente pode ser limitada.
- *QoS Offered*: estabelece os valores limites que o elemento de sofware suporta. A existência de uma restrição deste tipo significa que a qualidade oferecida deve ser garantida durante o funcionamento do software. Quando o fornecedor define uma restrição deste tipo, é ele quem deve garantir a qualidade no funcionamento do serviço, quando é o cliente que define a restrição, é ele quem deve garantir a qualidade na invocação do serviço.
- *QoS Contract*: representa os valores de qualidade finais, negociados e acordados entre o cliente e o fornecedor para a invocação de um serviço.

- *QoS Compound Constraint*: para a representação de restrições globais que podem ser decompostas em sub-restrições.

A Figura 7 mostra as metaclasses do modelo de restrições de QoS (OMG, 2006).

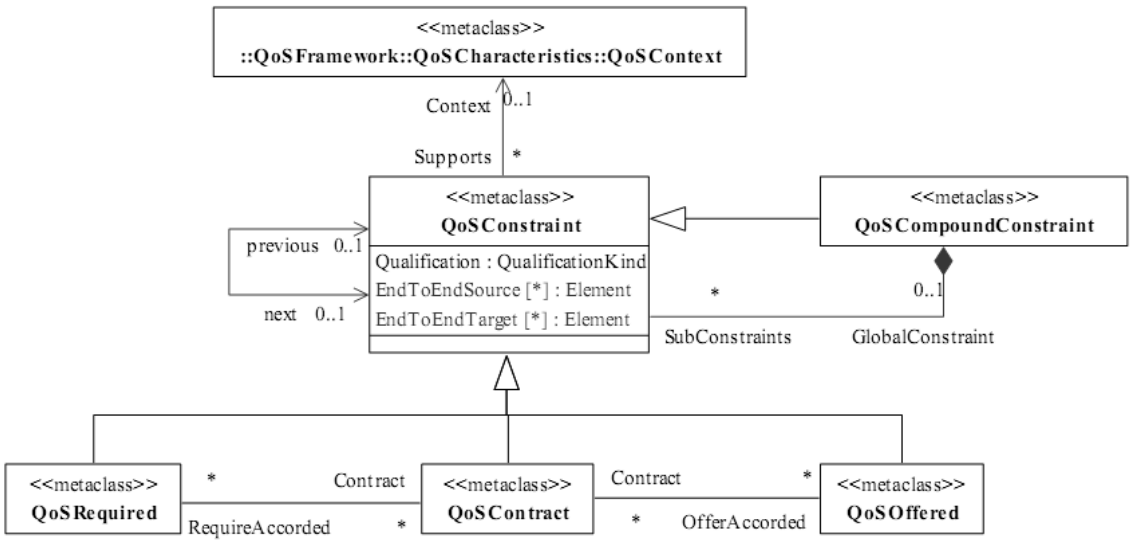

Figura 7: Modelo de restrições de QoS da OMG (2006)

3.2.3. O pacote *QoS Levels*

Os conceitos envolvidos neste pacote são:

- *QoS Level*: representa os diferentes modos de QoS que um subsistema suporta. Um nível de QoS descreve, através da propriedade `AllowedSpaces`, as condições para que o elemento de software possa trabalhar naquele nível e também os níveis de qualidade que são garantidos no nível. Quando os espaços requeridos não estão sendo mais respeitados, o software precisa começar a operar em outro nível, com garantias de qualidade diferentes. Para que a troca entre níveis ocorra, é necessário que exista uma transição de um nível para outro.
- *QoS Transition*: modela as transições permitidas entre os diferentes níveis de um elemento de software e as ações de adaptação que devem ser realizadas quando ocorre a transição.
- *QoS Compound Level*: inclui a definição de todos os níveis que definem o comportamento de qualidade de um elemento de software.

A Figura 8 mostra as metaclasses do modelo de níveis de QoS (OMG, 2006).

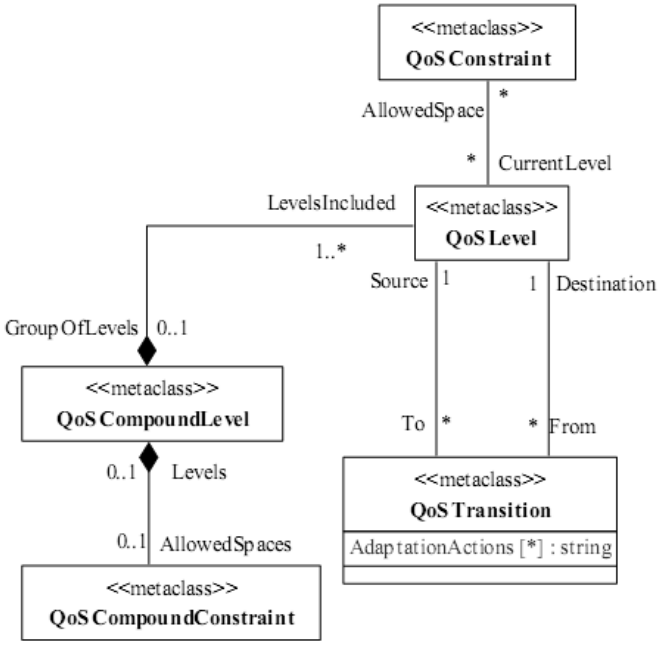

Figura 8: Modelo de níveis de QoS da OMG (2006)

3.3. Aplicações de QoS em Web Services Semânticos

A seguir, apresentaremos os principais trabalhos existentes no sentido de definir modelos para a especificação de QoS de Web Services Semânticos e descoberta de Web Services que satisfaçam a conjuntos definidos de requisitos de QoS, destacando também as limitações de cada abordagem.

3.3.1. A abordagem DAML-QoS

ZHOU et al. (2004) propõem um modelo para especificação de parâmetros e perfis de QoS, chamado DAML-QoS, e um algoritmo de inferência baseado em lógica descritiva para encontrar serviços Web que satisfaçam os critérios de busca especificados pelo usuário.

A ontologia DAML-QoS possui três camadas: a camada *QoS Profile* (Perfil de QoS), projetada para possibilitar a seleção de Web Services, a camada *QoS Property Definition* (definição de propriedades de QoS), para a elaboração das restrições de domínio e alcance das propriedades, e a camada *Metrics* (métricas) para a definição e a medição das métricas.

A camada *QoS Profile* define a classe básica `QoSProfile`, com três subclasses: `ProviderQoS`, que é a ontologia publicada pelo fornecedor do serviço mostrando a

qualidade garantida; `InquiryQoS`, especificada pelo cliente do serviço com os parâmetros requisitados; e `TemplateQoS`, que pode ser utilizada para criar uma base de um perfil de QoS para ser posteriormente estendido.

As propriedades de QoS, definidas na camada *QoS Property Definition*, são divididas em cinco tipos distintos: `QoSCore`, que é o tipo básico, utilizado quando uma propriedade serve tanto para entrada quanto para saída; `QoSInput` e `QoSOutput`, que são utilizados quando uma propriedade se aplica de forma diferente na entrada ou na saída do serviço; `QoSPrecondition`, que especifica as condições externas necessárias para que o serviço possa ser executado; e `QoSEffect`, que são os resultados causados pela execução do serviço. O alcance das propriedades de QoS é definido através das classes da camada *Metrics*.

As métricas de QoS, definidas na camada *Metrics*, são especificadas pela classe `Metric`, dividida nas subclasses `AtomicMetric` e `ComplexMetric`.

A associação entre um perfil de QoS e uma propriedade de QoS é feita no nível de classes e a definição do valor da métrica de QoS para a propriedade é feita através da cardinalidade desta associação. Isto é feito para facilitar a construção do algoritmo de busca, que utiliza um mecanismo de inferência para identificar a correspondência entre um perfil de QoS de serviço (`ProviderQoS`) e um de cliente (`InquiryQoS`).

O algoritmo de busca foi construído sobre um mecanismo de inferência em nível de classes (sistema RACER) e é capaz de identificar perfis equivalentes (fornecedor entrega exatamente o que o cliente deseja), ou perfis que englobam ou que possuem uma intersecção com outro. Todos os perfis de QoS que possuem intersecção com o perfil que se está buscando são retornados na consulta, classificados da seguinte forma:

- *Subsume*: o perfil buscado é um super-conceito do perfil encontrado, ou seja, o perfil encontrado possui todas as propriedades do perfil buscado, além de outras;
- *Exact*: o perfil encontrado e o perfil buscado são equivalentes;
- *PlugIn*: o perfil buscado é um sub-conceito do perfil encontrado, ou seja, o perfil encontrado possui apenas algumas das propriedades do perfil buscado;

- *Intersection*: o perfil encontrado e o perfil buscado têm algumas propriedades em comum;
- *Disjoint*: são os demais perfis, os que não têm nenhuma intersecção com o perfil buscado, ou seja, o perfil encontrado não tem nenhuma das propriedades do perfil buscado.

Nesta classificação, considera-se, em ordem, que os na categoria *Subsume* são os que atendem da melhor forma ao que se busca, e assim por diante, sendo os da categoria *Disjoint* os que absolutamente não atendem ao que se busca.

A Figura 9 mostra a definição de um perfil de QoS utilizando a ontologia DAML-QoS (ZHOU et al., 2004). Este perfil define que o tempo de resposta do serviço não pode ser maior que 20 segundos e o custo não pode ser maior que 1 dólar. O tempo de resposta é definido através da propriedade `ResponseTimeMSMetric` com uma cardinalidade de 2000 (milissegundos) e o custo é definido pela propriedade `CostUSCentMetric` com a cardinalidade de 100 (centavos de dólar).

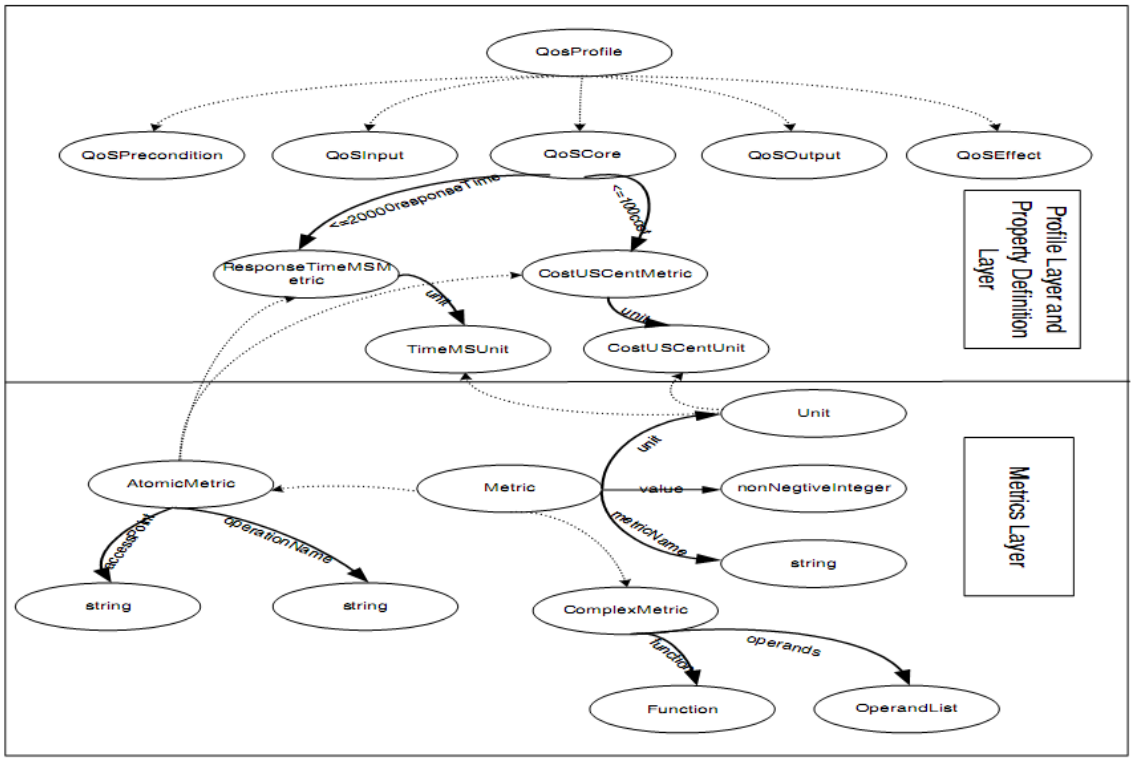

Figura 9: Exemplo de perfil de QoS definido em DAML-QoS (ZHOU et al., 2004)

A principal limitação da DAML-QoS consiste no fato de que a utilização de restrições de cardinalidade entre um perfil e as suas métricas de QoS para definir os valo-

res das características de QoS os limita a números inteiros não-negativos. Por causa disto, é preciso escolher com muito cuidado a unidade de medida das propriedades.

3.3.2. A abordagem QoSOnt

DOBSON et al. (2005) definem um modelo para especificação de QoS para serviços Web semânticos e uma solução para descoberta de serviços baseada em UDDI e XML. O modelo é proposto em três camadas: *Base QoS*, *Attribute* e *Usage Domain*.

A camada *Base QoS* (QoS base) define um conjunto mínimo de conceitos de QoS genéricos. A classe básica é `QoSAttribute`, que representa um atributo de QoS que pode ser mensurável ou não. Por isso existem duas subclasses de `QoSAttribute`: `MeasurableAttribute` e `UnmeasurableAttribute`. Todo atributo mensurável precisa estar ligado a uma métrica (classe `Metric`) que especifica como medir ou calcular o seu valor. As métricas possuem unidades, que são também especificadas na camada *Base QoS*. Por exemplo, a ontologia de unidades de tempo define todas as unidades de medida de tempo disponíveis e as conversões entre elas.

A camada *Attribute* (atributos) é usada para definir as propriedades de QoS que os serviços podem ter. Os autores apresentam no artigo taxonomias de atributos de confiabilidade e desempenho. Esta camada da QoSOnt pode ser estendida ou substituída para incluir a definição de outros domínios de atributos de QoS. A definição concreta de uma propriedade de serviços é obtida criando-se uma nova classe que será simultaneamente subclasse de alguma classe da camada *Base QoS* e de alguma classe definida nesta camada de atributos.

A camada *Usage Domain* (domínio de uso) é utilizada para conectar os atributos de QoS especificados com perfis de serviço descritos em qualquer formato, como, por exemplo, OWL-S.

A Figura 10 mostra um exemplo de definição de um atributo de QoS em QoSOnt e a ligação deste com um perfil de Serviço em OWL-S. A classe `servicePOSFOD` define um atributo e é simultaneamente subclasse de `ServiceQoSAttribute` (da camada base) e `ProbabilityOfSpecificFailureOnDemand` (da camada de atributos). O indivíduo `ephemerisPOSFOD` é uma instância da classe `servicePOSFOD` e é conectado como um parâmetro do serviço `ephemerisService`.

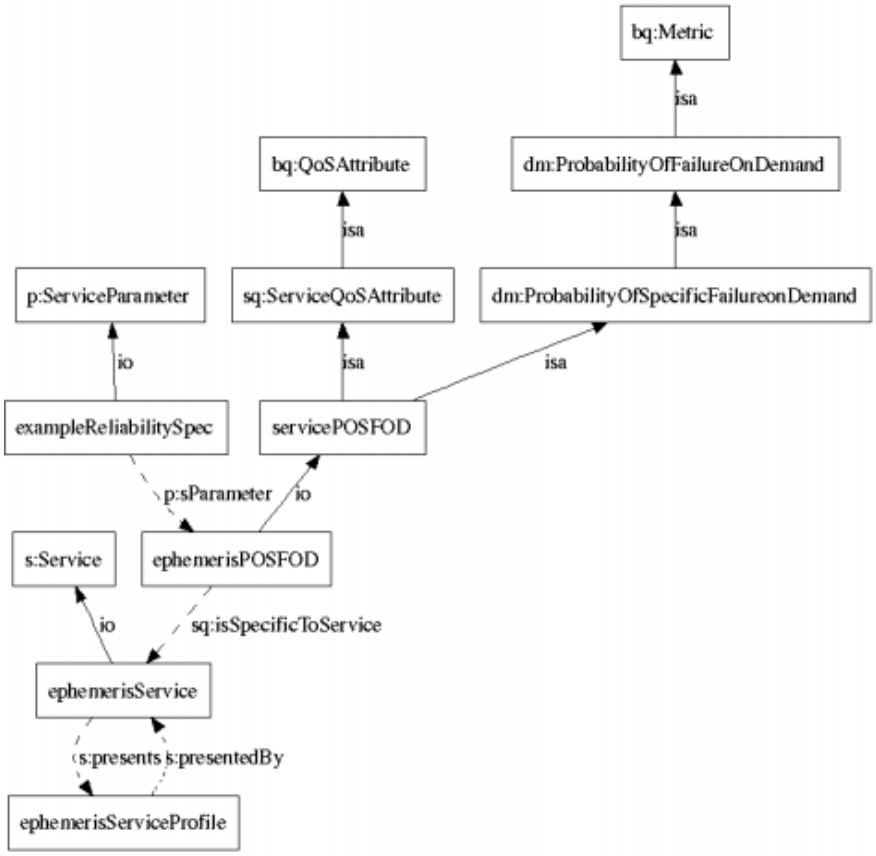

Figura 10: Exemplo de ligação QoSOnt/OWL-S (DOBSON et al., 2005)

O modelo proposto não provê meios para especificar um perfil de QoS de um Web Service com base em um conjunto de características de QoS, pois é preciso conectar cada atributo específico com o serviço. Também não é possível estender ou reutilizar uma especificação existente.

A descoberta de Web Services que atendam a determinados requisitos de QoS é feita através de um algoritmo que opera sobre UDDI. Primeiro, são recuperadas as descrições de Web Services que atendam aos requisitos funcionais desejados. As condições de QoS requeridas são definidas como expressões lógicas em formato XML. A partir disto, o algoritmo executa a expressão lógica sobre a descrição em QoSOnt de cada um dos Web Services listados e retorna aqueles cujo resultado da expressão seja verdadeiro. Pode-se notar que o algoritmo de descoberta não se utiliza de recursos específicos da ontologia ou de mecanismos de inferência: a ontologia é utilizada apenas como fonte de dados, porém a descoberta é feita através de execução de uma expressão lógica sobre os dados do perfil de cada serviço disponível.

3.3.3. A abordagem OWL-Q

KRITIKOS & PLEXOUSAKIS (2006) definiram a ontologia OWL-Q, que estende a OWL-S de modo a permitir a descrição de métricas e restrições de QoS e provê um algoritmo de *matching* semântico para descoberta de serviços. Utilizando essa ontologia é possível definir especificações de QoS completas, incluindo a possibilidade de especificar regras de conversão de unidades e métricas compostas.

A OWL-Q é dividida em várias *Facets* (partes), sendo que cada uma se concentra em uma parte da especificação de QoS e pode ser desenvolvida e estendida independentemente das demais. Um documento descrevendo um perfil de QoS completo deve fazer referências a todas as *Facets* da OWL-Q.

A *Connection Facet* (parte de conexão) conecta a OWL-Q com OWL-S e define os conceitos básicos para especificação de QoS. A conexão com OWL-S é feita através da classe `QoSAttribute`, que define um atributo de QoS e qual a métrica utilizada para medi-lo. Os atributos podem ser estáticos ou dinâmicos, assim como as métricas. A classe `QoSAttribute` é subclasse da classe `ServiceParameter` da OWL-S e referencia um `ServiceElement` da OWL-S, desta forma, atributos de QoS são definidos como parâmetros do perfil OWL-S do serviço para facilitar a conexão entre os modelos.

Na *Basic Facet* (parte básica), a especificação de QoS é feita através da classe `QoSSpec`, que tem as subclasses `QoSOffer`, para especificar as características garantidas pelo Web Service, e `QoSDemand`, para especificar as exigências do serviço. Um `ServiceProfile` é associado com múltiplos `QoSOffer` ou um `QoSRequest`. Um `QoSRequest` é dividido em um `QoSDemand` e um `QoSSelection`, que é uma lista de métricas e fatores de seleção.

A *QoS Metric Facet* (parte de métricas de QoS) descreve os elementos necessários para a especificação do modelo de métricas. Esta parte possui uma classe básica `QoSMetric`, que pode ser estendida para se criar as definições das diferentes métricas de QoS que serão utilizadas.

As *Function*, *Measurement Directive* e *Schedule Facets* (partes de funções, diretivas de medidas e agendamento) definem os conceitos para especificação de fórmulas e diretivas de medição de métricas, além da freqüência para computar uma métrica.

A *Unit Facet* (parte de unidades) descreve formalmente as unidades de medida, seus nomes, abreviações, sinônimos, sistemas e fatores de conversão.

A *QoSValueType Facet* (parte de tipos de valores de QoS) descreve os tipos de valores que uma métrica pode assumir, por exemplo: escalar, união, lista de valores, etc.

Para o processo de seleção, foi desenvolvido um algoritmo semântico que é capaz de identificar métricas equivalentes, mesmo se elas estiverem especificadas em ontologias separadas ou tiverem nomes diferentes. A identificação de equivalência é feita comparando as unidades, medidas e tipos de cada uma das métricas. Depois disto, as ofertas e demandas de QoS são transformadas em um CSP (*Constraint Satisfaction Problem* – problema de satisfação de restrições) (VAN HENTENRYCK & SARASWAT, 1996), considerando a informação de quais métricas são equivalentes. O CSP é resolvido por um mecanismo específico e os resultados são interpretados para apresentar os Web Services selecionados.

3.3.4. Ferramentas de busca de Web Services Semânticos com QoS

VU et al. (2006) propõem um *framework* para descoberta de Web Services semânticos considerando suas características de QoS. Eles utilizam o modelo WSMO para especificação de QoS e definem um mecanismo de classificação para indexar características de QoS e um algoritmo de descoberta de serviços.

A seleção de Web Services é feita através de um algoritmo que compara as restrições funcionais e não-funcionais especificadas pelo cliente com as descrições do Web Service para selecionar aqueles que atendem aos requisitos. Depois, é aplicado o algoritmo de classificação para ordenar os Web Services encontrados, identificando qual deles consegue atender melhor aos requisitos. Bastante ênfase é dada à otimização da consulta, apresentando técnicas para organizar e paralelizar as operações de consulta.

Os autores também apresentam uma abordagem baseada em P2P para a criação de um ambiente distribuído de publicação e descoberta de Web Services, adequado para ambientes de grande escala.

3.3.5. Limitações das abordagens existentes

Em nossa análise, a ontologia QoSOnt não possui a habilidade para especificar restrições complexas de QoS como mapeamento de dimensões ou perfis complexos de QoS. A ontologia DAML-QoS conta com esta característica, mas falha na especificação

dos valores das métricas de QoS, que ficam limitados a inteiros não-negativos em função da utilização de cardinalidades de associações para sua definição.

A ontologia OWL-Q é capaz de realizar a especificação completa dos perfis e requisitos de QoS. Entretanto, sua abordagem, assim como o *framework* proposto por VU et al., dependem de complexos algoritmos semânticos para encontrar especificações de QoS que preencham determinados requisitos, não utilizando, portanto, os padrões já estabelecidos mundialmente.

Podemos perceber, portanto, a inexistência de uma abordagem que seja capaz de permitir a descoberta de Web Services que atendam a requisitos específicos de QoS através de consultas simples às ontologias e utilizando os padrões já estabelecidos.

A Tabela 1 mostra a comparação entre as ontologias existentes para especificação semântica de QoS de Web Services Semânticos.

Tabela 1: Comparação das abordagens existentes de especificação semântica de QoS

	DAML-QoS	**QoSOnt**	**OWL-Q**
Associação de um perfil de QoS com um perfil de Web Service	Sim	Sim	Sim
Definição de um perfil de QoS como um conjunto de características de QoS	Sim	Não	Sim
Reuso e extensão de perfis de QoS	Sim	Não	Sim
Conversão entre unidades de medida	Sim	Sim	Sim
Dimensões compostas	Sim	Não	Sim
Mapeamento de dimensões	Não	Não	Sim
Flexibilidade na definição de valores	Não	Sim	Sim
Definição semântica de QoS oferecida e QoS requisitada	Sim	Não	Sim
Descoberta de QoS	Inferência DL	UDDI + XML + Lógica	Algoritmo semântico + CSP

4. A ONTOLOGIA QoS-MO

A QoS-MO (*Quality of Service Modeling Ontology* – Ontologia de Modelagem de Qualidade de Serviço) é uma ontologia descrita na linguagem OWL DL para a modelagem de características e restrições de QoS, que pode ser utilizada para a descrição de QoS de Web Services Semânticos ou Componentes de Software.

A QoS-MO define os conceitos relacionados ao domínio de QoS, porém a descrição individual de QoS de cada Componente não fica na QoS-MO. Assim como ocorre com OWL-S, é necessário importar a QoS-MO, estender as suas classes conforme a necessidade e definir os indivíduos (instâncias) que representam as características e restrições de QoS para criar a descrição de QoS de um Web Service ou Componente.

A ontologia QoS-MO foi criada primariamente para ser utilizada em conjunto com a ontologia OWL-S para a descrição funcional do Web Service. No entanto, QoS-MO é independente da linguagem de especificação funcional do Web Service ou Componente e pode ser utilizada com qualquer outra ontologia além da OWL-S, para descrever as características de QoS de qualquer tipo de Componente de Software.

Nossa opção pelo uso da OWL-S como padrão de ontologia para a descrição funcional de Web Services, ao invés de outros padrões disponíveis como, por exemplo, WSMO, deve-se ao fato de que a OWL-S já é um padrão reconhecido mundialmente, inclusive pela W3C, com grande possibilidade de aplicação prática.

O padrão de utilização da QoS-MO é a criação de uma ontologia para a especificação de QoS do Web Service através de QoS-MO, além da ontologia criada para a descrição funcional do Web Service através de OWL-S. No entanto, para situações mais simples, também é possível realizar a descrição funcional e não-funcional em uma só ontologia.

A Figura 11 ilustra a hierarquia de ontologias utilizadas para a especificação de QoS de um Web Service, utilizando ontologias separadas para a especificação funcional e não-funcional na Figura 11 (a) e utilizando apenas uma ontologia para ambas as descrições na Figura 11 (b).

Figura 11: Hierarquia de ontologias para especificação de QoS de Web Services

A ontologia QoS-MO teve como referência básica para sua definição o *Framework* para QoS da OMG (2006). Esta abordagem apresenta duas vantagens principais. A primeira é a adoção de um padrão bem maduro e que atende a todos os requisitos para uma especificação de QoS adequada. A segunda é que a QoS-MO terá uma correspondência muito próxima com o Profile UML de QoS da OMG, facilitando a tarefa de conversão automática de uma descrição baseada em QoS-MO para uma descrição baseada no Profile UML de QoS e vice-versa.

A ontologia QoS-MO contém três modelos que correspondem aos três submeta-modelos do meta-modelo de QoS da OMG (2006): o modelo ***QoS Characteristics***, que especifica as classes necessárias para descrição de características não funcionais de Web Services ou Componentes de Software, o modelo ***QoS Constraints***, que especifica as classes necessárias para definição de restrições e contratos de QoS, e o modelo ***QoS Levels*** que especifica as classes necessárias para descrever os modos e transições de QoS. A Figura 12 mostra a hierarquia destes modelos da QoS-MO.

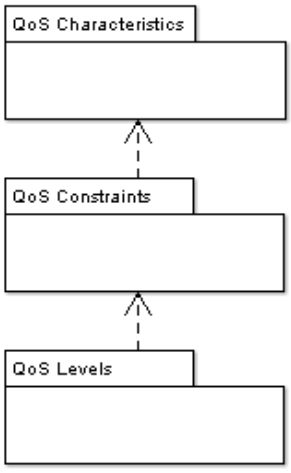

Figura 12: Hierarquia dos modelos da ontologia QoS-MO

A seguir, apresentaremos em detalhes cada uma das classes definidas nos três modelos da ontologia QoS-MO.

Como, ao falarmos em Componentes de Software, estamos incluindo também os Web Services, que são um tipo especial de Componente de Software, a partir deste momento, passaremos a nos referir apenas a Componentes de Software, ficando subentendido que a ontologia QoS-MO também pode ser utilizada para especificação de Web Services, sendo este, inclusive, seu principal objetivo.

4.1. O modelo QoS Characteristics

O modelo ***QoS Characteristics*** (Características de QoS) contém as classes para a especificação das características não funcionais dos Componentes de Software.

A Figura 13 mostra os elementos do modelo ***QoS Characteristics*** e as relações entre eles.

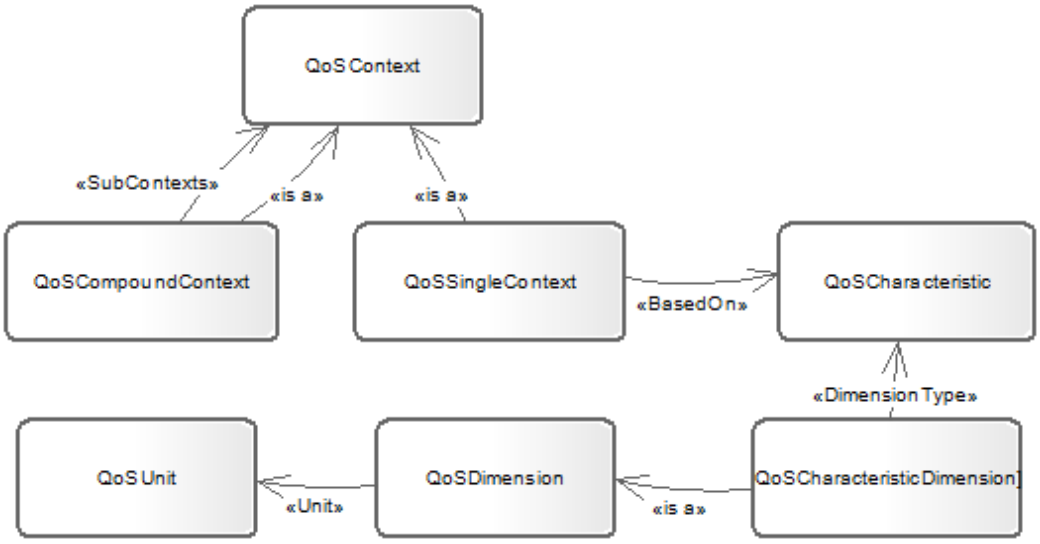

Figura 13: Elementos do modelo QoS Characteristics

4.1.1. `QoSCharacteristic`

A classe principal deste modelo é a classe **`QoSCharacteristic`**. Uma instância desta classe corresponde a uma característica não funcional e mensurável de um serviço. Sendo assim, cada subclasse de `QoSCharacteristic` define um tipo de característica não funcional que pode ser aplicado a cada serviço.

Exemplos de características não funcionais de serviços geralmente adotadas são:

latência, *throughput* (capacidade de fluxo), disponibilidade, confiabilidade, precisão e segurança. As características não funcionais de um serviço também podem corresponder a características específicas do domínio em questão.

A classe `QoSCharacteristic` pode utilizar mecanismos de reutilização, extensão e especialização.

O mecanismo de extensão/especialização é realizado através da criação de subclasses de uma classe derivada de `QoSCharacteristic` , o que permite a definição de novos tipos de características não funcionais que aproveitam a estrutura do tipo de característica estendido e acrescentam novas funcionalidades à característica.

O mecanismo de reutilização é realizado através da inclusão de uma determinada característica como uma dimensão de outra. Isto é feito com a criação de dimensões que se referem a outras características, através da classe `QoSDimension`.

A Tabela 2 lista as propriedades da classe `QoSCharacteristic`.

Tabela 2: Propriedades da classe `QoSCharacteristic`

Propriedade	Tipo	Descrição
`isInvariant`	`boolean`	Indica se os valores das dimensões da característica podem ser atualizados dinamicamente.

4.1.2. `QoSDimension`

Uma característica pode ter várias formas diferentes de medição, ou podem ser necessárias diversas medidas para quantificar adequadamente uma característica. As dimensões para a mensuração das características são modeladas usando a classe **`QoSDimension`**.

Exemplos de dimensões para a característica latência são: latência mínima, latência máxima e *jitter* (flutuação) (OMG, 2006).

A classe `QoSDimension` não tem um relacionamento pré-definido com a classe `QoSCharacteristic`. Ao invés disto, cada subclasse de `QoSCharacteristic`, representando uma característica não funcional, deve definir as suas dimensões através de novas propriedades de objeto cujo alcance seja uma subclasse de `QoSDimension`. Isto fará com que as dimensões sejam propriedades da característica.

Esta estratégia é a mesma utilizada pelo Perfil UML da OMG.

A Tabela 3 lista as propriedades da classe `QoSDimension`.

Tabela 3: Propriedades da classe `QoSDimension`

Propriedade	Tipo	Descrição

`direction`	Enumeração:
	`Increasing`
	`Decreasing`
		Indica se os valores são ordenados de forma crescente ou decrescente.

`Unit`	`QoSUnit`	Especifica a unidade de medida.

`statisticalQualifier`	Enumeração:
	`Maximum`
	`Minimum`
	`Range`
	`Mean`
	`Average`
	`Variance`
		Identifica o tipo de qualificador estatístico (por exemplo, média, mediana, desvio padrão, etc.) quando a dimensão representar um valor estatístico.

4.1.3. `QoSCharacteristicDimension` (subclasse de `QoSDimension`)

Já foi dito que uma `QoSCharacteristic` pode ter outra `QoSCharacteristic` como dimensão para possibilitar o reuso de características.

Para que isto seja possível, uma subclasse de `QoSDimension` pode ser definida como uma referência a uma subclasse de `QoSCharacteristic`. Este tipo de dimensão é criado a partir da classe `QoSCharacteristicDimension`, que é uma subclasse de `QoSDimension` com uma propriedade adicional para definir a característica na qual se baseia a dimensão definida.

A Tabela 4 lista as propriedades da classe `QoSCharacteristicDimension`.

Tabela 4: Propriedades da classe `QoSCharacteristicDimension`

Propriedade	Tipo	Descrição
`DimensionType`	`QoSCharacteristic`	Identifica a característica a qual a dimensão faz referência.

4.1.4. `QoSContext` e subclasses

A classe **`QoSContext`** permite a definição de expressões de QoS que combinem múltiplas características. Um contexto simples, definido pela subclasse **`QoSSingleContext`**, referencia uma ou mais Características de QoS. Um contexto

composto, definido pela subclasse **QoSCompoundContext**, permite que um contexto seja composto por outros QoSContext.

A Tabela 5 lista as propriedades da classe QoSSingleContext.

A Tabela 6 lista as propriedades da classe QoSCompoundContext.

Tabela 5: Propriedades da classe QoSSingleContext

Propriedade	Tipo	Descrição
BasedOn	QoSCharacteristic	Identifica a característica ou as características que compõem o contexto de QoS definido.

Tabela 6: Propriedades da classe QoSCompoundContext

Propriedade	Tipo	Descrição
SubContexts	QoSContext	Identifica os subcontextos de QoS que compõem o contexto definido.

4.1.5. QoSUnit

A classe **QoSUnit** é utilizada para a definição da Unidade de Medida de uma dimensão. A Tabela 7 lista as propriedades da classe QoSUnit.

Tabela 7: Propriedades da classe QoSUnit

Propriedade	Tipo	Descrição
name	String	Nome da unidade de medida.
abbreviation	String	Representação abreviada da unidade de medida.

4.1.6. QoSValue

A classe QoSDimension poderia ter uma propriedade value para permitir a definição dos valores das dimensões quando forem criadas as instâncias. No entanto, é mais interessante definir o valor da dimensão separadamente da definição da dimensão, já que algumas dimensões podem ter valores dinâmicos.

Para conseguir isto, foi definida uma classe simples chamada **QoSValue** que contém apenas a propriedade **value**. Desta forma, um indivíduo da classe QoSDimension que precise definir um valor deve ser simultaneamente indivíduo da classe QoSValue.

A Tabela 8 lista as propriedades da classe QoSValue.

Tabela 8: Propriedades da classe `QoSValue`

Propriedade	Tipo	Descrição
`value`	`(qualquer)`	Valor medido ou definido para a dimensão.

4.1.7. `QoSDimensionMapping`

Como é possível definir várias formas diferentes de medir uma mesma característica, através da definição de diferentes dimensões, pode ocorrer muitas vezes que a especificação da qualidade necessária seja feita em dimensões diferentes da especificação da qualidade garantida pelo serviço.

Por exemplo, um determinado Web Service pode definir seus parâmetros de qualidade para a característica tempo de resposta do serviço que fornece através de três dimensões: tempo médio de processamento da requisição (para interpretar os dados recebidos), tempo médio de execução (para realizar o serviço solicitado) e tempo médio de envio da resposta (para gerar e disponibilizar para envio os dados de resposta). No entanto, um cliente pode definir como requisito que o serviço que irá invocar deve ter simplesmente um determinado tempo médio de resposta (tempo desde o início do recebimento da requisição até a disponibilização da resposta para envio). Se não houver uma forma de mapear o valor da dimensão definida pelo cliente com os valores das dimensões definidas pelo fornecedor, será impossível combinar estes serviços.

Na QoS-MO, scripts de mapeamento entre dimensões podem ser definidos através da classe **`QoSDimensionMapping`**. Todo mapeamento precisa ter a identificação das dimensões envolvidas, através da identificação das dimensões que originam o mapeamento e das dimensões que são o destino do mapeamento. Esta identificação é realizada pelas propriedades **`sourceDimension`** e **`targetDimension`** da classe `QoSDimensionMapping`, respectivamente.

O uso da classe `QoSDimensionMapping` permite que mecanismos automáticos de busca ou de negociação de parâmetros de QoS identifiquem uma dimensão mesmo que o seu valor não tenha sido explicitamente definido para uma característica, desde que exista um script de mapeamento entre esta dimensão e outras dimensões da mesma característica ou até de características diferentes.

A fórmula utilizada para o cálculo do valor convertido das dimensões envolvidas no mapeamento é definida através da propriedade **`expression`** da classe `QoSDimensionMapping`. Os scripts de mapeamento são definidos utilizando expressões lógicas e aritméticas e expressões condicionais (if/else). Todas as operações são escritas utilizando a mesma sintaxe aceita por C, C++ e Java.

A Tabela 9 lista as propriedades da classe `QoSDimensionMapping`.

Tabela 9: Propriedades da classe `QoSDimensionMapping`

Propriedade	Tipo	Descrição
`sourceDimension`	`QoSDimension`	A dimensão ou as dimensões de origem do mapeamento.
`targetDimension`	`QoSDimension`	A dimensão de destino do mapeamento.
`Expression`	`String`	A expressão que define um script de mapeamento, permitindo calcular o valor para a dimensão destino a partir dos valores das dimensões de origem.

No exemplo de mapeamento de tempo de resposta apresentado anteriormente, suponhamos que dimensões definidas pelo Web Service fornecedor fossem representadas pelas classes `AverageRequestTime`, `AverageExecutionTime` e `AverageReplyTime`, e a dimensão definida pelo Web Service cliente fosse representada pela classe `AverageResponseTime`.

A Figura 14 mostra as classes que estariam envolvidas em um mapeamento das dimensões deste exemplo. Para realizá-lo, deve ser criada uma subclasse da classe `QoSDimensionMapping`. Nesta subclasse, é definida uma restrição para a propriedade `sourceDimension` definindo que ela aceita indivíduos do conjunto das classes `Average_RequestTime_Dimension`, `Average_ExecutionTime_Dimension` e `Average_ReplyTime_Dimension` e uma restrição para a propriedade `targetDimension` definindo que ela aceita indivíduos da classe `Average_ResponseTime_Dimension`.

A propriedade `expression` também teria uma restrição que definiria o script para calcular o valor da dimensão destino:

```
Average_ResponseTime_Dimension = Average_RequestTime_Dimension +
Average_ExecutionTime_Dimension + Average_ReplyTime_Dimension;
```

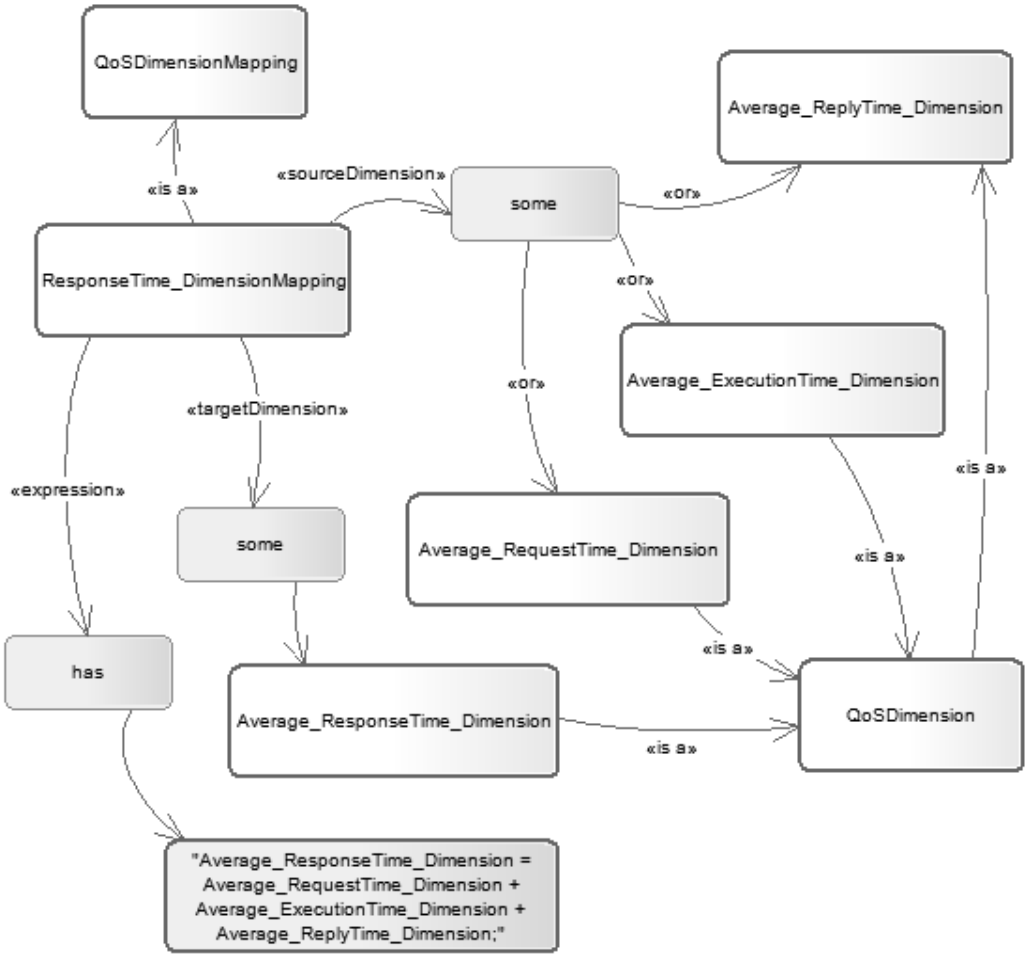

Figura 14: Exemplo de um Mapeamento de Dimensões

Após a criação das classes de mapeamento de dimensões na ontologia, a aplicação dos mapeamentos às características existentes na ontologia – ou seja, a descoberta das dimensões que podem ser inferidas a partir das existentes e o cálculo do valor das dimensões inferidas – deve ser feita de forma automática.

Nosso projeto prevê que, no momento da inclusão de características descritivas de um novo Componente, automaticamente podem ser criadas as instâncias das dimensões que podem ser descobertas através de mapeamento.

O motivo para que a realização desta descoberta de características mapeadas seja na inclusão de características, e não no momento da pesquisa somente, é que esta operação envolve a execução de scripts de cálculo, não podendo ser resolvida apenas com inferência ou consultas SPARQL. Como a inclusão de cada característica ocorre apenas uma vez, mas a pesquisa por esta característica ocorrerá várias vezes, é mais interessan-

te já realizar o mapeamento das características na inclusão, criando na ontologia instâncias de `QoSDimension` que correspondam às características mapeadas com os valores já pré-calculados.

Para realizar todo este trabalho de mapeamento de dimensões, uma ferramenta de suporte ao cadastramento de Componentes deverá ser desenvolvida. No capítulo 7 serão apresentadas maiores informações sobre trabalhos futuros.

4.1.8. Exemplo de definição de Características de QoS

A Figura 15 mostra um exemplo de definição de características de QoS e de um perfil de QoS para o Web Service BravoAir, encontrado nos documentos de exemplo da OWL-S (OWL-S, 2004).

BravoAir é um serviço fictício de venda de passagens aéreas, que foi descrito utilizando as ontologias OWL-S para exemplificar o seu uso. As ontologias do exemplo BravoAir definem um serviço chamado `BravoAir_ReservationAgent`, descrito pelo perfil `Profile_BravoAir_ReservationAgent`.

Segundo a descrição do perfil, encontrada na ontologia, o serviço "provê reserva de vôos baseado na especificação de uma requisição de vôo. Esta tipicamente envolve um aeroporto de partida, um aeroporto de destino, uma data de partida e, se uma passagem de retorno é necessária, uma data de retorno. Se o vôo com as especificações desejadas estiver disponível, um itinerário e um número de reserva serão retornados."

Em nosso exemplo, as características disponíveis na ontologia são representadas pelas classes **`AvailabilityCharacteristic`**, que tem uma dimensão – `MeanAvailability` – e **`ResponseTimeCharacteristic`**, que tem duas dimensões – `MaxResponseTime` e `AverageResponseTime`. Como já foi dito, a relação entre as classes das características e as das dimensões é feita através da criação de propriedades das classes das características que têm como alcance cada uma das dimensões.

O serviço BravoAir possui um contexto de QoS representado pela instância **`BravoAirQoS`** (instância de `QoSSingleContext`) que contém duas características, representadas pelas instâncias **`BravoAir_ResponseTime`** (instância de `ResponseTimeCharacteristic`) e **`BravoAir_Availability`** (instância de `AvailabilityCharacteristic`).

A característica `BravoAir_ResponseTime` teve informada a dimensão `MaxResponseTime` através da instância `BravoAir_MaxResponseTime`, com valor de no máximo 5 segundos de tempo de resposta, e a característica `BravoAir_Availability` teve informada a dimensão `MeanAvailability` através da instância `BravoAir_MeanAvailability`, com valor de 99% de disponibilidade.

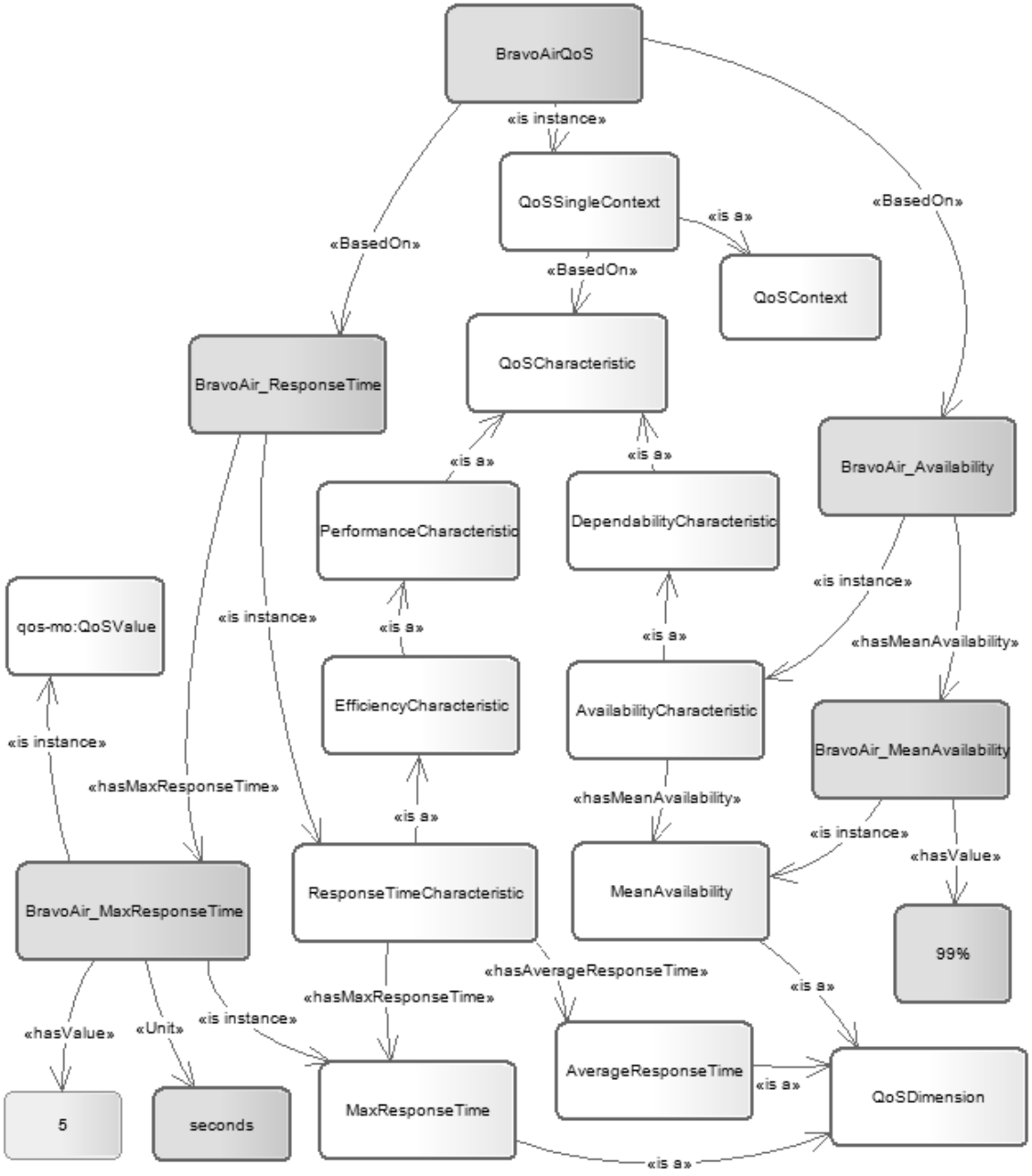

Figura 15: Exemplo de definição de um perfil de QoS

4.2. O modelo *QoS Constraints*

As classes do modelo *QoS Characteristics* são capazes de identificar e definir precisamente quais são as características não funcionais que podem ser especificadas para cada Componente de Software e fazer relações e mapeamentos entre elas. O próximo passo é definir conjuntos de restrições de QoS que denotem as capacidades apresentadas por um Componente em particular. Isto é feito através das classes do modelo ***QoS Constraints*** (Restrições de QoS).

A Figura 16 mostra os elementos do modelo ***QoS Constraints*** e as relações entre eles.

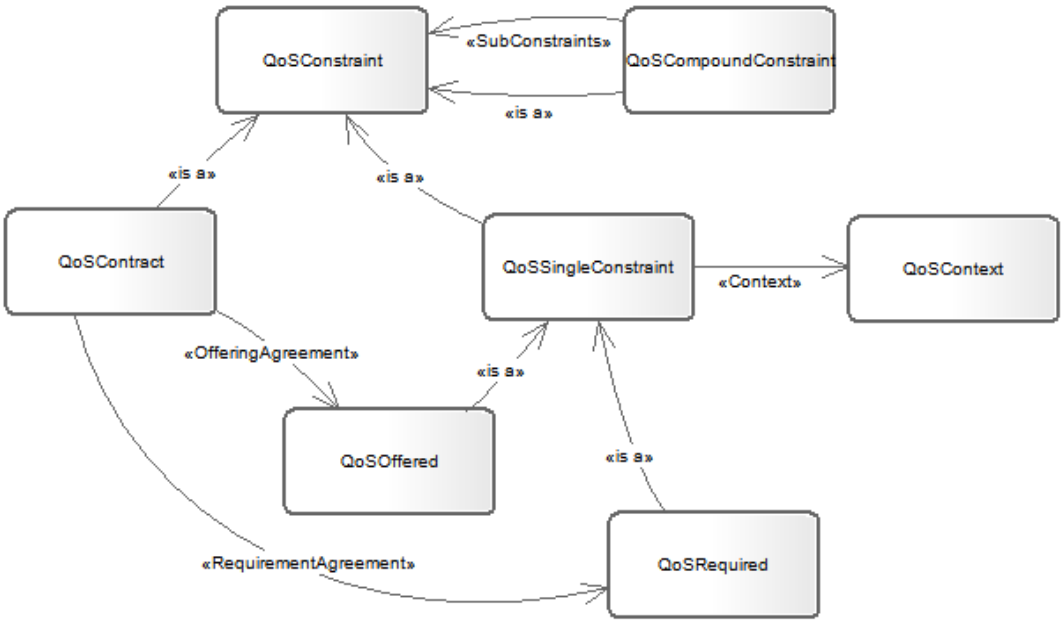

Figura 16: Elementos do modelo *QoS Constraints*

Existem três tipos de restrições de QoS: qualidade oferecida (*QoS Offered*), qualidade requisitada (*QoS Required*) e contrato de QoS (*QoS Contract*). As restrições oferecidas e requisitadas ganham um significado diferente, dependendo se são especificadas para um fornecedor de um serviço ou para um solicitador de um serviço.

4.2.1. `QoSConstraint`

A classe **`QoSConstraint`** é a superclasse para todas as restrições de QoS. Ela não possui propriedades e não deve ser instanciada diretamente, servindo apenas para agrupar todas as subclasses de restrições de QoS.

4.2.2. `QoSContract` (subclasse de `QoSConstraint`)

Contratos de QoS são representados por instâncias da classe **`QoSContract`**. Um contrato de QoS representa o nível de qualidade que efetivamente é acordado por dois serviços em uma composição e que será efetivamente garantido em tempo de execução.

Quando uma composição entre Web Services ou Componentes é feita estaticamente e todas as características envolvidas possuem valores estáticos, é possível definir também estaticamente o contrato de QoS envolvido. Quando algumas das características envolvidas têm seu valor definido apenas dinamicamente ou quando seu valor depende dos recursos disponíveis, o valor final do contrato de QoS só poderá ser obtido durante a execução. Mesmo em um caso como este, é possível identificar estaticamente quais são as características envolvidas no contrato e quais são os valores limite para as características, baseando-se nas restrições oferecidas e requisitadas pelos serviços.

A definição de Contratos de QoS vai além da simples especificação de qualidade oferecida pelo serviço, permitindo o estabelecimento de acordos de nível de serviço (SLA) entre as partes envolvidas, através da definição das garantias que cada parte oferecerá na composição.

Apesar de ser possível considerar que a definição de SLAs extrapola a especificação de qualidade de serviço, que é o objetivo principal da QoS-MO, este recurso também está incluído na ontologia para torná-la mais completa, oferecendo maiores possibilidades de aplicação.

A Tabela 10 lista as propriedades da classe `QoSContract`.

Tabela 10: Propriedades da classe `QoSContract`

Propriedade	Tipo	Descrição
`Offering Agreement`	`QoSOffered`	Os parâmetros de qualidade de serviço que serão garantidos pelas partes envolvidas no Contrato.
`Requirement Agreement`	`QoSRequired`	Os parâmetros de qualidade de serviço que foram requisitados pelas partes envolvidas no Contrato.

4.2.3. `QoSSingleConstraint` (subclasse de `QoSConstraint`)

As restrições de QoS oferecidas ou requisitadas especificam as características de

qualidade referenciando os contextos (`QoSContext`) que forneçam as expressões de QoS e valores que serão associados à restrição.

A classe **`QoSSingleConstraint`** é utilizada para agrupar estes dois tipos de restrições que, apesar de terem semântica diferente, possuem as mesmas propriedades.

A Tabela 11 lista as propriedades da classe `QoSSingleConstraint`.

Tabela 11: Propriedades da classe `QoSSingleConstraint`

Propriedade	Tipo	Descrição
`Context`	`QoSContext`	O Contexto ou os Contextos de QoS que definem os parâmetros de QoS que fazem parte da restrição.

`constraint Qualification`	Enumeração:
	`Guarantee`
	`BestEffort`
	`ThesholdBestEffort`
	`CompulsoryBestEffort`
	`None`
		Os parâmetros de qualidade de serviço que foram requisitados pelas partes envolvidas no Contrato.

4.2.4. `QoSOffered` (subclasse de `QoSSingleConstraint`)

As restrições de qualidade de serviço oferecida são representadas por instâncias da classe **`QoSOffered`**.

Quando um fornecedor de um serviço estabelece uma restrição deste tipo, ele define qual é a qualidade que será garantida durante a execução do serviço invocado pelo cliente. Esta é a definição de qualidade de serviço em seu sentido mais simples e a base para o funcionamento dos mecanismos de qualidade de serviço.

Quando um cliente de um serviço estabelece uma restrição deste tipo, ele define quais são as garantias que serão mantidas quando estiver invocando o serviço. O cliente pode garantir, por exemplo, que não tentará enviar mais requisições simultâneas do que o fornecedor do serviço é capaz de processar. Este tipo de restrição não está incluído no que seria considerado propriamente uma definição de QoS, mas pode ser utilizado para a especificação de acordos SLA através dos Contratos de QoS, como já mencionado.

A classe `QoSOffered` não acrescenta nenhuma propriedade à classe `QoSSingleConstraint`.

4.2.5. `QoSRequired` (subclasse de `QoSSingleConstraint`)

As restrições de qualidade de serviço requisitada são representadas por instâncias da classe **`QoSRequired`**.

Quando um cliente de um serviço estabelece uma restrição deste tipo, ele define que o fornecedor deve garantir um determinado nível de qualidade no fornecimento do serviço para que este possa ser utilizado pelo cliente sem problemas. Esta definição é utilizada em conjunto com a definição da qualidade oferecida pelo serviço para verificar se o serviço atende às necessidades do cliente.

Quando um fornecedor de um serviço estabelece uma restrição deste tipo, ele define qual é a qualidade que o cliente deve garantir na invocação do serviço para obter a qualidade esperada. Em outras palavras, o cumprimento, por parte do cliente, dos requisitos especificados na restrição `QoSRequired` do fornecedor do serviço é uma pré-condição para que o fornecedor seja capaz de garantir a qualidade especificada na sua definição `QoSOffered`. Por exemplo, o fornecedor pode especificar qual é o número máximo de requisições simultâneas que ele pode receber do cliente para poder garantir os níveis de qualidade oferecidos. Este tipo de restrição pode ser utilizado em conjunto com a definição de qualidade oferecida pelo cliente e, assim como aquela, não está incluído no que seria considerado propriamente uma definição de QoS, mas pode ser utilizado para a especificação de acordos SLA através dos Contratos de QoS.

Uma restrição de qualidade requisitada também pode ser definida por um mecanismo de busca de Componentes a partir dos requisitos definidos pelo usuário para a realização das buscas. A restrição `QoSRequired` criada pode, então, ser comparada com as restrições `QoSOffered` dos serviços que estão sendo pesquisados para eliminar aqueles que não são capazes de cumprir com a qualidade desejada, ou classificá-los segundo um critério que seja capaz de definir qual atende melhor os critérios desejados.

A classe `QoSRequired` não acrescenta nenhuma propriedade à classe `QoSSingleConstraint`.

4.2.6. `QoSCompoundConstraint` (subclasse de `QoSConstraint`)

Uma restrição de QoS também pode ser composta em diversas subrestrições. Neste caso, a classe **`QoSCompoundConstraint`** é utilizada para definir a restrição global

que conterá as demais subrestrições. A Tabela 12 lista as propriedades da classe `QoSCompoundConstraint`.

Tabela 12: Propriedades da classe `QoSCompoundConstraint`

Propriedade	Tipo	Descrição
`SubConstraints`	`QoSConstraint`	Identifica as restrições de QoS que compõem a restrição definida.

4.2.7. Exemplo de definição de Restrições de QoS

A Figura 17 exemplifica a definição de restrições de QoS do serviço BravoAir.

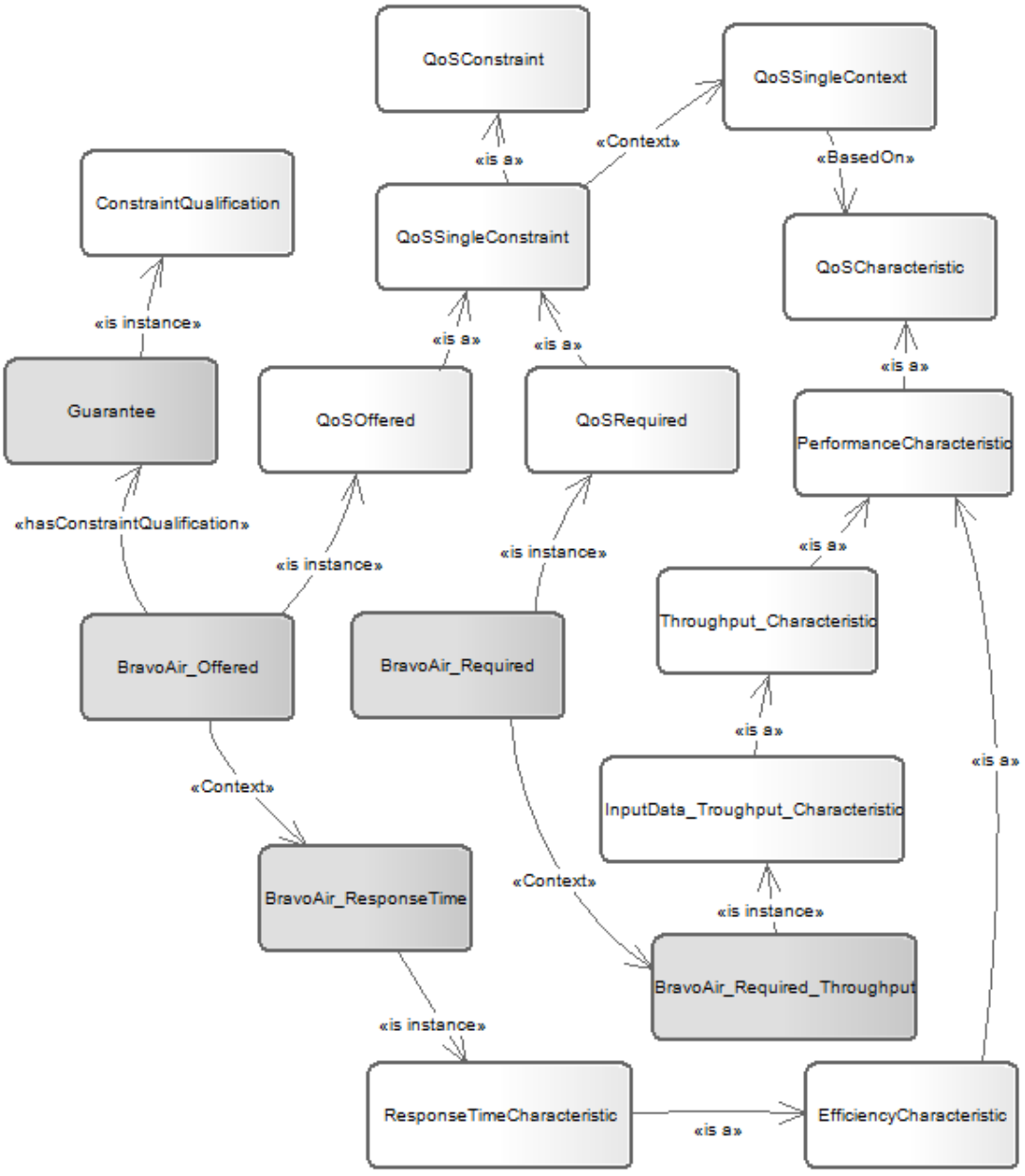

Figura 17: Exemplo de definição de restrições de QoS

Uma restrição de QoS oferecida é definida pela instância **`BravoAir_Offered`** (instância de `QoSOffered`). Esta restrição define o contexto `BravoAir_ResponseTime` e a qualificação `Guarantee`, ou seja, ela define que o serviço BravoAir garante o tempo de resposta especificado através das dimensões da característica `BravoAir_ResponseTime`, por exemplo, um tempo de no máximo 5 segundos (conforme ilustrado na Figura 15).

Também é definida uma restrição de QoS requisitada pela instância **`BravoAir_Required`** (instância de `QoSRequired`). Esta restrição define o contexto `BravoAir_Required_Throughput`, indicando que a qualidade de serviço garantida só é alcançada se for respeitado o nível de fluxo de entrada de dados especificado pelas dimensões da característica `BravoAir_Required_Throughput`. Por exemplo, poderia ser definido um máximo de 100 requisições por segundo para que seja garantido o tempo de resposta especificado.

4.3. O modelo *QoS Levels*

Em alguns casos, os níveis de qualidade de serviço suportados por um Componente de Software podem depender de variáveis do ambiente de execução ou dos recursos disponíveis. O modelo ***QoS Levels*** (Níveis de QoS) é utilizado para especificar os diferentes níveis de QoS que um Componente suporta.

A Figura 18 mostra os elementos do modelo ***QoS Levels*** e as relações entre eles.

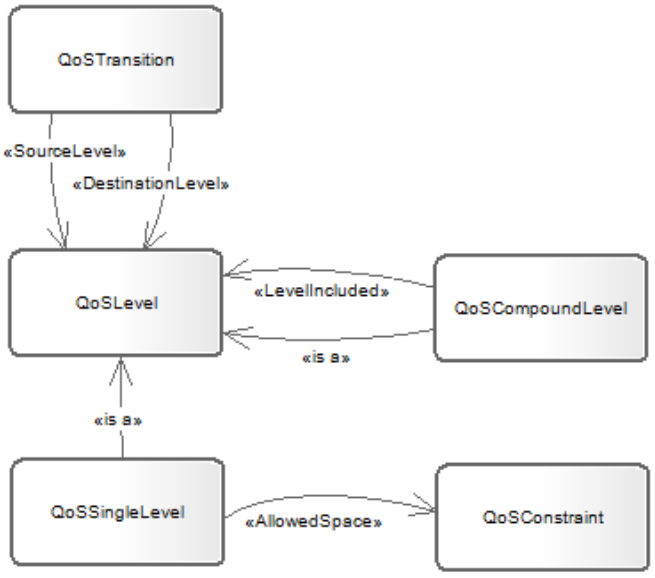

Figura 18: Elementos do modelo *QoS Levels*

4.3.1. `QoSLevel` e subclasses

Cada nível suportado por um Componente é representado por uma instância da classe **`QoSLevel`**.

Um nível simples é composto por várias restrições de QoS e representado por uma instância da classe **`QoSSingleLevel`**. As restrições do tipo `QoSRequired` do nível identificam quais são as condições necessárias para que o serviço possa ser executado naquele nível, enquanto as restrições do tipo `QoSOffered` identificam as restrições de qualidade que serão garantidas naquele nível.

Também é possível representar um nível composto por vários subníveis, através da classe **`QoSCompoundLevel`**.

A Tabela 13 lista as propriedades da classe `QoSSingleLevel`.

A Tabela 14 lista as propriedades da classe `QoSCompoundLevel`.

Tabela 13: Propriedades da classe `QoSSingleLevel`

Propriedade	Tipo	Descrição
`AllowedSpaces`	`QoSContext`	Identifica os parâmetros de QoS que são garantidos ou requisitados quando o serviço está operando no nível especificado.

Tabela 14: Propriedades da classe `QoSCompoundLevel`

Propriedade	Tipo	Descrição
`LevelsIncluded`	`QoSLevel`	Identifica os subníveis de QoS que compõem o nível definido.

4.3.2. `QoSTransition`

Quando um Componente não é mais capaz de operar em um nível determinado é necessário haver uma transição para outro nível. Um exemplo deste tipo de ocorrência poderia ser um Web Service que começou a receber mais requisições do que anteriormente e, desta forma, terá que aumentar o seu tempo máximo de resposta.

As transições entre níveis de um serviço são representadas pela classe **`QoSTransition`**. Quando uma transição ocorrer, provavelmente os serviços envolvidos nas composições terão que renegociar seus parâmetros de QoS e criar novos

`QoSContract` para representar os novos níveis acordados, revendo dinamicamente seus SLAs.

Uma subclasse de `QoSTransition` pode especificar todas as ações que sejam necessárias para adaptar o serviço ao novo nível.

A Tabela 15 lista as propriedades da classe `QoSTransition`.

Tabela 15: Propriedades da classe `QoSTransition`

Propriedade	Tipo	Descrição
`SourceLevel`	`QoSLevel`	Identifica o nível de QoS a partir do qual ocorre a transição de nível.
`DestinationLevel`	`QoSLevel`	Identifica o nível de QoS para o qual o serviço irá após a transição.
`AdaptationActions`	`String`	Especificação das ações de adaptação necessárias para completar a transição. Esta especificação dependerá de um ambiente de execução que possa entender e executar as ações definidas.

4.3.3. Exemplo de definição de Níveis de QoS

A Figura 19 mostra um exemplo de definição de níveis de QoS para os serviço BravoAir.

Dois níveis de QoS são definidos. O nível **`BravoAir_NormalLevel`** está associado com as restrições de QoS `BravoAir_Offered` e `BravoAir_Required` que definem, através das características `BravoAir_Required_Throughput` e `BravoAir_ResponseTime`, que o serviço garante um tempo de resposta máximo de 5 segundos, desde que o fluxo de entrada de requisições seja no máximo 100 instruções por segundo.

Já o nível **`QoS_CriticalLevel`** está associado com as restrições de QoS `BravoAir_CriticalOffered` e `BravoAir_CriticalRequired` que definem, através das características `BravoAir_Critical_ResponseTime` e `BravoAir_Critical_Throughput` que o serviço faz o melhor esforço para tentar manter o tempo de resposta máximo de 10 segundos caso o fluxo de entrada de requisições seja no máximo 1.000 instruções por segundo.

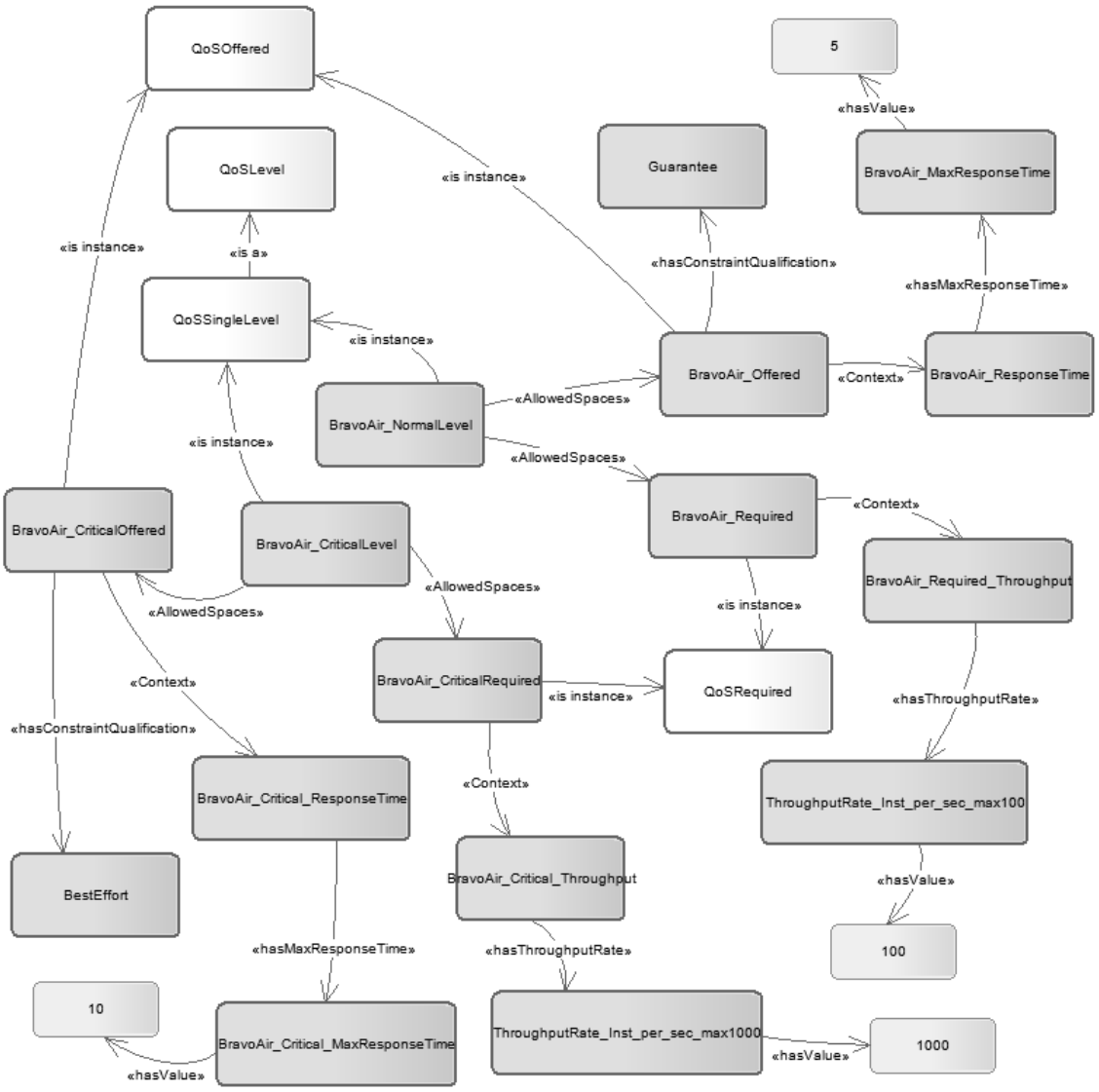

Figura 19: Exemplo de definição de níveis de QoS

4.4. Extensão da OWL-S

A ontologia QoS-MO pode ser utilizada para estender a ontologia OWL-S com restrições de QoS.

As restrições de QoS especificadas com QoS-MO podem se referir a um serviço OWL-S como um todo ou apenas a um processo específico de um serviço. Sendo assim, uma restrição de QoS (indivíduos da classe `QoSConstraint`) poderão ser associados a serviços (indivíduos da classe **`Profile`** da OWL-S) ou a processos (indivíduos da classe **`Process`** da OWL-S).

As ofertas e requisições de QoS sempre fazem referência a um serviço ou processo, portanto, indivíduos das classes `QoSOffered` e `QoSRequired` sempre serão associados a apenas uma instância de `Profile` ou `Process`. Já o contrato de QoS é sempre definido entre dois serviços, portanto, uma instância de `QoSContract` será associada com duas instâncias de `Profile` ou duas instâncias de `Process` (de serviços diferentes). A associação entre as restrições de QoS e os perfis ou modelos de Serviço são feitas pela classe **`QoSProfile`** da QoS-MO.

A Figura 20 mostra os elementos da QoS-MO e da OWL-S envolvidos na extensão da OWL-S.

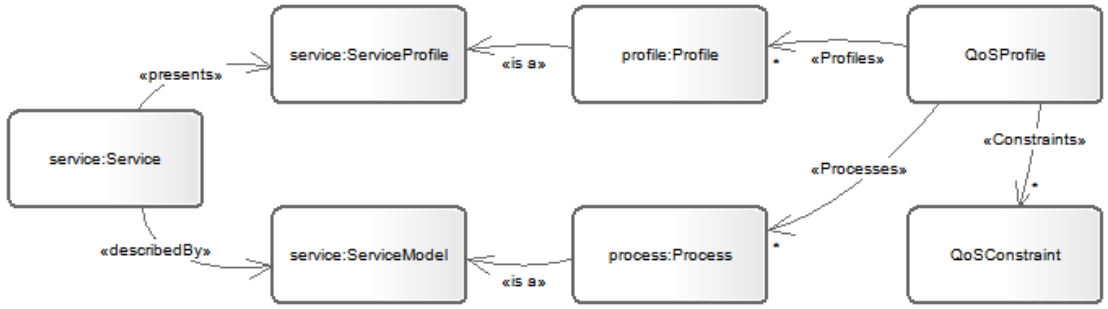

Figura 20: Elementos do modelo para extensão da OWL-S

4.4.1. Exemplo de extensão de um perfil OWL-S

A Figura 21 mostra um exemplo de definição de perfil de QoS associada com a definição OWL-S do Web Service BravoAir, representado pelo indivíduo **`Profile_BravoAir_ReservationAgent`**.

A ontologia exemplo do OWL-S que define o serviço de Agente de Reservas BravoAir especifica os serviços em uma hierarquia que os classifica. Assim, o perfil de serviço `Profile_BravoAir_ReservationAgent` é uma instância da hierarquia `AirlineTicketing`, que, por sua vez, é um subtipo da hierarquia `E_Commerce`.

Aqui, é definida a restrição de QoS oferecida pelo serviço, através do indivíduo **`BravoAir_Offered`** (instância de `QoSOffered`), composto por um contexto de QoS representado pelo indivíduo **`BravoAirQoS`** (instância de `QoSCompoundContext`). A conexão do perfil de QoS do serviço com o perfil funcional do Serviço é feito através do indivíduo **`BravoAir_QoSProfile`** (instância de `QoSProfile`).

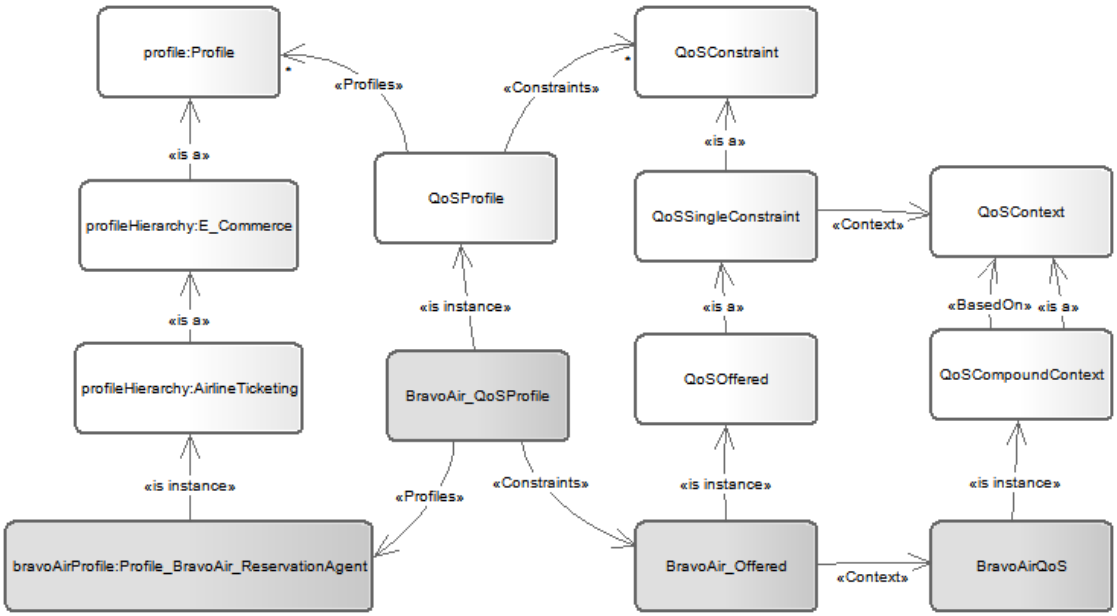

Figura 21: Exemplo completo da especificação de QoS do Web Service BravoAir

5. O MECANISMO DE BUSCA

A especificação semântica de QoS utilizando a ontologia QoS-MO pode ser utilizada para encontrar automaticamente Componentes de Software que atendam a um determinado conjunto de requisitos de QoS. Os requisitos procurados podem ser especificados por um usuário, através de uma ferramenta de busca manual de Componentes, ou podem ser obtidos através da especificação de requisitos de QoS exigidos por determinado Componente e processados por ferramentas automáticas de negociação de QoS.

Para tornar possível este tipo de busca sobre a ontologia QoS-MO, mantendo, ao mesmo tempo, uma implementação simples e eficiente, construímos um mecanismo de busca que demonstra que é possível realizar isto através da geração de consultas utilizando a linguagem SPARQL (W3C, 2007).

Nosso mecanismo combina eficazmente os recursos de inferência OWL, para avaliar a transitividade das relações de composição e especialização entre características, contextos, restrições e níveis de QoS, e os recursos da linguagem SPARQL para encontrar as restrições e perfis de QoS que atendem aos requisitos especificados.

O mecanismo de busca consiste em uma API (*Application Programming Interface* – Interface de Programação de Aplicações), que permite identificar as características de QoS disponíveis na ontologia que descreve os Serviços ou Componentes disponíveis – baseada na ontologia QoS-MO –, executar consultas para localizar Serviços ou Componentes que obedeçam a um conjunto de requisitos de QoS e retornar uma lista dos Componentes de Software correspondentes.

5.1. A API de Busca

A API de busca foi desenvolvida na linguagem Java, utilizando as APIs Jena e ARQ (JENA, 2008). Sendo assim, pode ser utilizada diretamente por qualquer aplicação também desenvolvida em Java. Além disto, foi previsto o desenvolvimento de uma interface que expõe a API através de um Web Service, fazendo com que ela possa ser utilizada também por aplicações desenvolvidas em qualquer outra linguagem que suporte a utilização de Web Services.

A API é subdividida em três grupos funcionais e possui ainda uma interface central que implementa uma Fábrica para a criação dos elementos acessados pelo cliente de cada um dos três grupos funcionais.

O primeiro grupo, que denominamos **Descoberta de Características**, é composto por mecanismos que efetuam a leitura da árvore de características de QoS disponíveis na ontologia de descrição de serviços e as retorna ao cliente. Com isso, é possível construir um suporte para negociação automática de QoS, que possa identificar dinamicamente as características disponíveis no sistema. Tais mecanismos também tornam possível a construção de uma interface gráfica para que o usuário possa escolher, dentre as características de QoS, quais ele deseja que estejam presentes nos Web Services ou Componentes que irá utilizar. Essa funcionalidade é demonstrada na interface Web de busca, que será descrita na seção 5.2.

O segundo grupo funcional, denominado **Execução de Consultas**, permite que o cliente especifique uma consulta e a execute. A execução de consultas é realizada por um componente que converte os requisitos de QoS especificados pelo cliente em uma consulta SPARQL e executa a consulta resultante sobre a ontologia que contém a descrição da qualidade de serviço suportada pelos Web Services ou Componentes disponíveis.

O terceiro e último grupo, denominado **Recuperação de Componentes**, é formado por um componente que recupera todos os Componentes de Software que satisfazem a consulta e os retorna, na forma de uma lista, ao cliente. Esta funcionalidade prevê a possibilidade de definição de múltiplas formas de recuperação de componentes, dependendo da ontologia utilizada para descrevê-los. Uma implementação padrão para recuperar descrições de serviços em OWL-S é fornecida e outras implementações específicas podem ainda ser desenvolvidas.

A Figura 22 apresenta o esquema de utilização da API de Busca sobre QoS-MO: a API, invocada por um cliente, lê os dados solicitados em uma ontologia com informações de QoS baseada na ontologia QoS-MO e retorna os dados solicitados ao cliente. A Figura 23 mostra, de forma simplificada, as seqüências de interação entre o Cliente, a interface da API e os seus grupos funcionais.

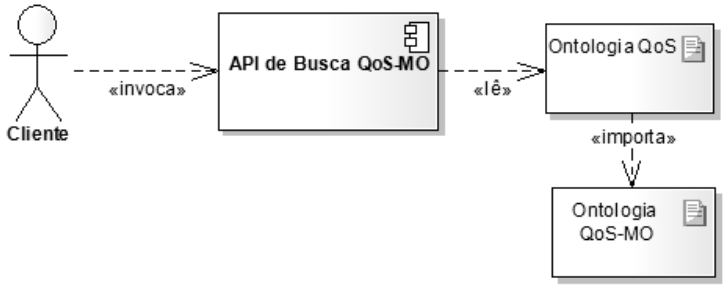

Figura 22: Esquema de utilização da API de Busca QoS-MO

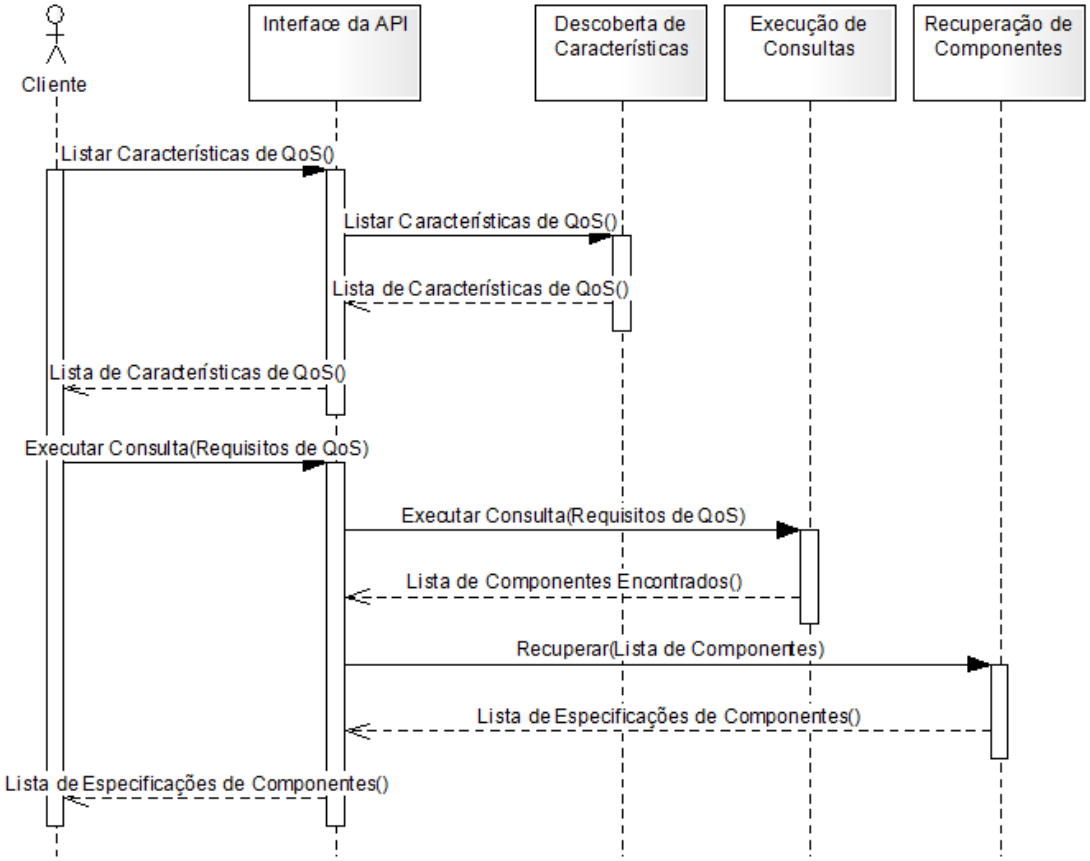

Figura 23: Seqüência de utilização da API de busca

A seguir, apresentaremos os detalhes de construção do método de leitura da ontologia utilizado pela API e de cada um dos grupos funcionais da API.

5.1.1. Leitura da Ontologia

A leitura e interpretação da Ontologia de descrição de QoS dos Componentes é feita através da criação de um modelo OWL em memória da API Jena.

No momento da leitura da Ontologia, é necessário utilizar um mecanismo de inferência para avaliar a transitividade das hierarquias e das composições entre as classes, já

que a consulta SPARQL não é capaz de realizar nenhuma inferência. A realização de todo o processo de inferência na carga da ontologia simplifica posteriormente o processo da consulta, já que a expressão SPARQL pode simplesmente encontrar os indivíduos que atendem aos critérios especificados, que já estarão previamente classificados através das asserções e inferências presentes no modelo ontológico.

Para habilitar a realização das consultas necessárias sobre as ontologias especificadas a partir da QoS-MO, existe a necessidade de avaliar a transitividade das seguintes relações:

Tabela 16: Inferências de relações necessárias para o mecanismo de busca

Relação inferida	**Utilização**
Hierarquia de subclasses de `QoSCharacteristic`	Esta regra de inferência irá identificar, para cada indivíduo de característica de QoS presente na ontologia, todas as superclasses de tipos de características existentes. Isto permitirá que as consultas possam especificar um tipo de característica e encontrar qualquer perfil de QoS que inclua qualquer característica que seja subclasse da especificada na consulta.
Composição de Contextos (`QoSCompoundContext SubContexts QoSContext`)	Esta regra irá identificar, para cada Contexto de QoS composto, quais são todas as características de QoS presentes, não importando quantos níveis de composição existirem (um contexto pode ser composto por outros contextos compostos e assim por diante). Isto permitirá que a consulta encontre perfis de QoS cuja restrição de QoS que define a característica consultada esteja em um contexto composto.
Composição de Restrições (`QoSCompoundConstraint SubConstraints QoSConstraint`)	Esta regra irá identificar, para cada Restrição de QoS composta, quais são todas as sub-restrições presentes, de forma semelhante à composição de contextos. Isto permitirá que a consulta encontre perfis de QoS cuja característica desejada se encontre em uma restrição composta.
Composição de Níveis (`QoSCompoundLevel LevesIncluded QoSLevel`)	Esta regra irá identificar, para cada nível de QoS composto, quais são todos os sub-níveis presentes, de forma semelhante às composições de contextos e restrições. Isto permitirá que a consulta encontre perfis de QoS cuja restrição que define a característica desejada está definida em um nível composto.

A API Jena implementa um mecanismo de inferência sobre RDFS e três mecanismos de inferência sobre OWL, sendo que cada um vai crescendo em complexidade e capacidade e é capaz de inferir tudo que o anterior consegue, acrescentando ainda mais regras ao mecanismo de inferência.

Para nosso mecanismo de busca, há apenas dois tipos de construções que precisam ser inferidos para garantir que as relações listadas na Tabela 16 sejam avaliadas: a

construção `rdfs:subClassOf`, para a definição de hierarquias de classes, e a construção `owl:TransitiveProperty`, para a avaliação da transitividade das propriedades que definem as composições entre as classes da Ontologia.

Para conseguir o resultado desejado, utilizamos o mecanismo de inferência OWL mais básico da API Jena, denominado ***Micro OWL***, que é o mecanismo de inferência OWL mais rápido da API e que é capaz de realizar as inferências necessárias para nosso mecanismo de busca. O mecanismo de inferência *Micro OWL* suporta as seguintes construções (JENA, 2008):

- `rdf:type`,
- `rdfs:subClassOf`,
- `rdfs:subPropertyOf`,
- `rdfs:domain`,
- `rdfs:range`,
- `owl:intersectionOf`,
- `owl:unionOf` (parcial),
- `owl:equivalentClass`,
- `owl:Thing`,
- `owl:equivalentProperty`,
- `owl:inverseOf`,
- `owl:FunctionalProperty`,
- `owl:InverseFunctionalProperty`,
- `owl:SymmetricProperty`,
- `owl:TransitiveProperty`,
- `owl:hasValue`.

5.1.2. Descoberta de Características

A descoberta de características é um componente simples da API que gera uma árvore contendo a hierarquia das características de QoS disponíveis na ontologia de domínio.

Geralmente, a descoberta de características é utilizada para a geração de toda a árvore de características existente na ontologia. Além disto, também é possível obter uma sub-árvore de características de QoS de uma determinada categoria ou uma sub-árvore de características que são sub-tipos de alguma outra.

Outra função que também faz parte da descoberta de características é a listagem das dimensões das características. As listagens de dimensões também são importantes para que o cliente da API possa montar o requisito de QoS utilizado para a consulta. A

API pode listar todas as dimensões existentes na ontologia, ou pode listar as dimensões específicas de uma característica particular ou de um conjunto de características definidas por uma superclasse comum.

A implementação da API de descoberta de características utiliza a API Jena para obter a definição das classes `QoSCharacteristic` da ontologia de domínio, que são mapeadas para objetos da classe `QoSCharacteristicClass` do modelo da API. Depois disto ela vai, recursivamente, obtendo as subclasses de cada uma delas e inserindo características filhas. A lista resultante é então retornada ao cliente.

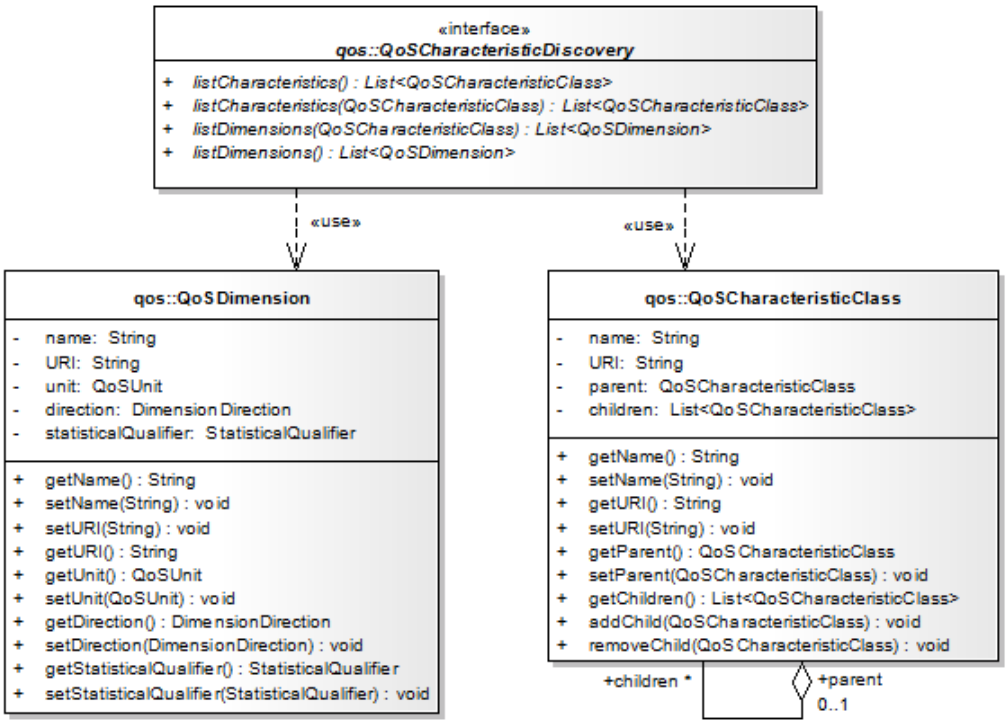

Figura 24: Interface de descoberta de características da API de busca

A Figura 24 mostra a parte da interface da API referente à função de descoberta de características. O cliente da API irá acessar esta função através da interface `QoSCharacteristicDiscovery` que contém métodos para retornar uma lista de `QoSCharacteristicClass` das características que fazem parte da ontologia ou da sub-árvore selecionada. As características listadas, por sua vez, podem conter características filhas através do relacionamento *parent-children*.

A Tabela 17 lista os métodos definidos na interface de descoberta de características (`QoSCharacteristicDiscovery`).

Tabela 17: Métodos da interface de descoberta de características

Método	Retorno	Descrição
`listCharacteristics()`	`List<QoSCharacteristic>`	Lista todas as características existentes na ontologia.
`listCharacteristics (QoSCharacteristicClass)`	`List<QoSCharacteristic>`	Lista a hierarquia de características que são filhas da característica especificada.
`listDimensions (QoSCharacteristicClass)`	`List<QoSDimension>`	Lista as dimensões que estão vinculadas a todas as características do tipo especificado.
`listDimensions()`	`List<QoSDimension>`	Lista todas as dimensões existentes na ontologia.

5.1.3. Execução de Consultas

A execução de consultas é o componente da API que cria e executa as consultas SPARQL sobre a ontologia de descrição de QoS dos Web Services ou Componentes. Como a inferência das especializações de características e composições de contextos, restrições e níveis já foi feita no momento da leitura da Ontologia, consultas SPARQL simples são capazes de encontrar perfis de QoS que contenham determinadas características de QoS conforme definidas na consulta do usuário. Esta abordagem é capaz de realizar as buscas desejadas sem a necessidade de desenvolvimento de um novo algoritmo complexo.

Para buscar Serviços ou Componentes que atendam a determinados requisitos de QoS, o cliente primeiro cria um Consulta de QoS, em seguida executa esta consulta para obter a lista das restrições de QoS que atendem aos requisitos especificados e, finalmente, pode solicitar a listagem dos perfis de QoS que contém as restrições encontradas.

A Figura 25 mostra os elementos da interface de execução de consultas da API. O cliente acessa este componente da API através da interface `QoSQueryExecution` e também da interface `QoSQuery`, que é utilizada para definir as Consultas de QoS.

As consultas de QoS, definidas através da interface `QoSQuery`, consistem em uma lista de requisitos de QoS definidos através de instâncias da classe `QoSRequirement`.

Cada requisito de QoS é definido por um conjunto de quatro atributos: uma característica de QoS (`characteristic`), uma dimensão (`dimension`), um operador

(`operator`) e um valor (`value`), que são interpretados da seguinte forma: buscar um perfil de QoS que defina uma característica `characteristic` que contenha uma dimensão `dimension`, cujo valor respeite o valor `value` considerado o operador `operator`. Os operadores disponíveis são: = (igual), < (menor), > (maior), <= (menor ou igual), >= (maior ou igual), <> (diferente).

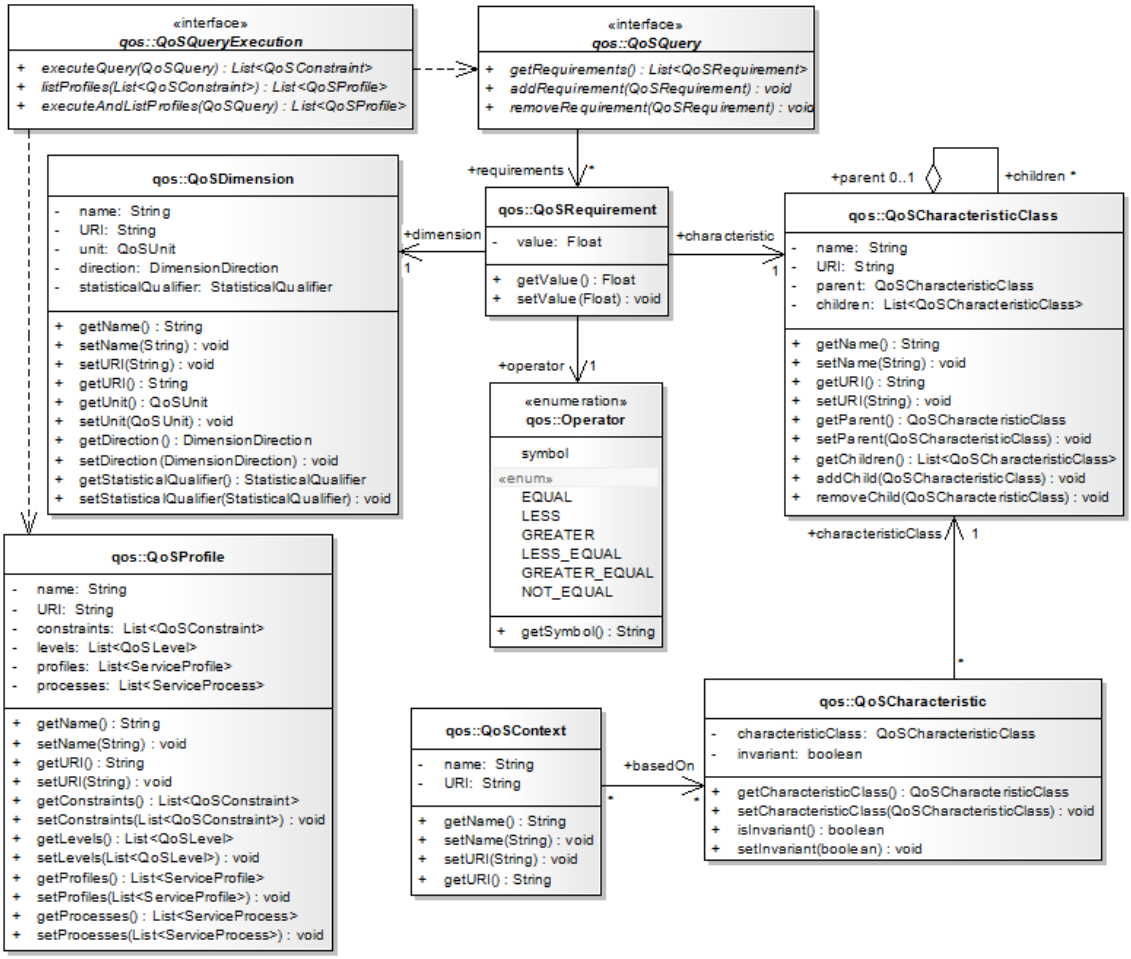

Figura 25: Interface de execução de consultas da API de busca

A Tabela 18 lista os métodos definidos na interface de execução de consultas (`QoSQueryExecution`).

Tabela 18: Métodos da interface de execução de consultas

Método	Retorno	Descrição
`executeQuery(QoSQuery)`	`List<QoSConstraint>`	Executa a consulta especificada, listando todas as restrições de QoS ofertado que atendam aos requisitos.

`listProfiles (List<QoSConstraint>)`	`List<QoSProfile>`	Executa uma consulta para listar os perfis de QoS que contêm as restrições de QoS especificadas.
`executeAndListProfiles (QoSQuery)`	`List<QoSProfile>`	Combina os dois métodos anteriores: executa a consulta especificada, já listando todos os perfis de QoS cujas restrições atendam aos requisitos.

5.1.4. Geração das consultas SPARQL

O método `executeQuery` da interface de execução de consultas, que é o método que sempre é chamado para a execução de consultas, invoca a implementação do algoritmo que gera a consulta SPARQL correspondente aos requisitos especificados no objeto `QoSQuery` e, em seguida, executa a consulta gerada. A consulta é criada a partir do seguinte modelo:

```
1  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
   PREFIX qos-mo: <http://www.inf.ufsc.br/~fortes/Ontology/QoS-MO#>
2  SELECT DISTINCT ?constraint
   WHERE {
3   {
     ?charact rdf:type <{URI da característica}>.
     ?charact ?predicate ?dimension.
     ?predicate rdfs:range <{URI da dimensão}>.
     ?dimension qos-mo:value ?value.
     FILTER (?value {operador} {valor})
    }
4   {
     ?context rdf:type qos-mo:QoSContext.
     ?context qos-mo:BasedOn ?charact.
    }
    UNION
    {
     ?context qos-mo:SubContexts ?subcontext.
     ?subcontext rdf:type qos-mo:QoSContext.
     ?subcontext qos-mo:BasedOn ?charact.
    }
5   {
     ?constraint rdf:type qos-mo:QoSOffered.
     ?constraint qos-mo:Context ?context.
    }
    UNION
    {
     ?constraint qos-mo:SubConstraints ?subconstraint.
     ?subconstraint rdf:type qos-mo:QoSOffered.
     ?subconstraint qos-mo:Context ?context.
    }
6  }
```

Neste modelo de consulta SPARQL, os campos marcados entre chaves {} são substituídos, respectivamente, pelo URI da característica especificada na consulta, o URI da dimensão especificado, o símbolo do operador especificado e o valor especificado. A consulta é dividida nas seguintes partes:

1	Definição dos prefixos utilizados na consulta.
2	Abertura da consulta e definição do campo projetado.
3	Este grafo encontra as características que atendem ao requisito de pesquisa, ou seja, que contém a dimensão especificada e o valor dentro da faixa especificada.
4	Estes dois grafos especificam a seleção dos Contextos de QoS que possuem as características selecionados pela parte 3 da consulta. O primeiro grafo da união encontra os contextos que referenciam diretamente as características encontradas. O segundo grafo da união encontra os contextos compostos que possuem sub-contextos que referenciam as características encontradas.
5	Estes dois grafos especificam a seleção das Restrições de QoS, do tipo QoS ofertado, que possuem os contextos selecionados pela parte 4 da consulta. O primeiro grafo da união encontra as restrições que referenciam diretamente os contextos encontrados. O segundo grafo da união encontra as restrições compostas que possuem sub-restrições que referenciam os contextos encontrados.
6	Fechamento da consulta

Quando há mais de um requisito de pesquisa, é preciso repetir a consulta para cada um deles. Há dois tipos de consulta com mais de um requisito:

- Atendimento total: neste tipo de consulta, são retornados apenas os perfis de QoS que atendam totalmente a todos os requisitos de QoS especificados. Neste caso, é executada uma consulta para cada requisito e, ao final, é retornada a intersecção dos resultados encontrados.
- Atendimento parcial: neste tipo de consulta, são retornados todos os perfis de QoS que atendam a pelo menos um dos requisitos de QoS especificados. Neste caso, é executada uma consulta para cada requisito e, ao final, é retornada a união dos resultados encontrados.

O método `listProfiles` da interface de execução de consultas obtém os perfis de QoS que incluem cada uma das restrições de QoS retornadas pelo método `executeQuery`. Para isto, é gerada e executada a seguinte consulta para cada uma das restrições:

1	`PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>` `PREFIX qos-mo: <http://www.inf.ufsc.br/~fortes/Ontology/QoS-MO#>`
2	`SELECT ?qosprofile` `WHERE {`

3	`{`
	`?qosprofile rdf:type qos-mo:QoSProfile.`
	`?qosprofile qos-mo:Constraints <{URI da restrição}>.`
	`}`

4	UNION
	{
	?qosprofile rdf:type qos-mo:QoSProfile.
	?qosprofile qos-mo:Levels ?qoslevel.
	?qoslevel qos-mo:AllowedSpaces {URI da restrição}>.
	}

5	UNION
	{
	?qosprofile rdf:type qos-mo:QoSProfile.
	?qosprofile qos-mo:Levels ?qoslevel.
	?qoslevel qos-mo:LevelsIncluded ?sublevel.
	?sublevel qos-mo:AllowedSpaces {URI da restrição}>.
	}

6	}

Neste modelo de consulta SPARQL, os campos marcados entre chaves {} são substituídos pelo URI da restrição cujos perfis devem ser buscados. A consulta é dividida nas seguintes partes:

1	Definição dos prefixos utilizados na consulta
2	Abertura da consulta e definição do campo projetado
3	Grafo que seleciona os perfis de QoS que se relacionam diretamente com a restrição de QoS especificada.
4	União de grafo que seleciona os perfis de QoS que possuem níveis de QoS que se relacionam diretamente com a restrição de QoS especificada.
5	União de grafo que seleciona os perfis de QoS que possuem níveis compostos, e pelo menos um dos sub-níveis se relaciona com a restrição de QoS especificada.
6	Fechamento da consulta

Cada execução desta consulta gera uma lista dos perfis de QoS correspondentes a uma das restrições listadas. O algoritmo do método que as executa vai reunindo todos os perfis de QoS encontrados em uma só lista, para retorná-la ao cliente da API.

5.1.5. Recuperação de Componentes

A recuperação de componentes é utilizada para ler todas as informações disponíveis na ontologia sobre um determinado componente, a partir de seu perfil de QoS. Esta parte da API é utilizada para que seja possível recuperar todas as informações sobre um componente encontrado pelo mecanismo de execução de consultas.

A recuperação de componentes é invocada através de uma interface básica, porém sua implementação é dependente da ontologia utilizada para a descrição funcional do Componente. Uma implementação padrão, para a recuperação de perfis de Web Services descritos em OWL-S, está pronta. Novas implementações podem ser escritas facilmente, bastando escrever uma classe Java que leia as informações do Componente da

ontologia de descrição funcional carregada no modelo OWL Jena, e criar um objeto da classe `Service` descrevendo o perfil lido.

A Figura 26 mostra os elementos do modelo de recuperação de componentes da API. A recuperação é invocada através da interface `QoSComponentRetriever` que, utilizando a implementação previamente configurada (ou a implementação de recuperação de Web Services OWL-S por padrão, se não houver nenhuma configuração), lê os perfis completos de especificação do componente e os retorna em um objeto da classe `Service`.

O perfil de QoS também é completamente lido, a partir do URI que fora previamente encontrado pelo mecanismo de execução de consultas, e um objeto da classe `QoSProfile` é preenchido com as informações do perfil de QoS incluindo as informações de níveis e restrições.

Figura 26: Interface de recuperação de componentes da API de busca

A Tabela 19 lista os métodos definidos na interface de recuperação de componentes (`QoSComponentRetriever`).

Tabela 19: Métodos da interface de recuperação de componentes

Método	Retorno	Descrição
`listServices (List<QoSProfile>)`	`List<Service>`	Executa uma consulta para encontrar todos os perfis funcionam que descrevem os Componentes que possuem os perfis de QoS retornados pela execução de consultas de QoS. Esta consulta apenas identifica os perfis funcionais, sem recuperar todas as informações do perfil.
`retrieveService Information(Service)`	`Service`	Recupera a especificação completa, funcional e não-funcional, de um Componente e preenche os objetos correspondentes com as informações lidas da ontologia.

5.2. A interface Web de busca

Uma interface Web de busca foi desenvolvida para testar e validar o funcionamento da API de busca.

Esta pequena aplicação Web é capaz de apresentar as características disponíveis na ontologia de especificação de QoS para o usuário, utilizando os mecanismos de descoberta de características de QoS da API de busca.

Com base nessa informação, o usuário é capaz de especificar uma consulta empregando a interface Web, mostrada na Figura 27.

Após a construção da consulta, que pode conter diversas condições, o botão de busca (Search) ativa os mecanismos de execução da consulta e de listagem de Serviços Web providos pela API.

Cada condição de busca criada na interface Web é convertida pela própria aplicação Web em um requisito de QoS, especificado por um objeto da classe `QoSRequirement`. A lista de requisitos é, então, utilizada como entrada para os méto-

dos do componente de execução de consultas da API para recuperar os perfis de QoS que atendam aos requisitos especificados pelo usuário.

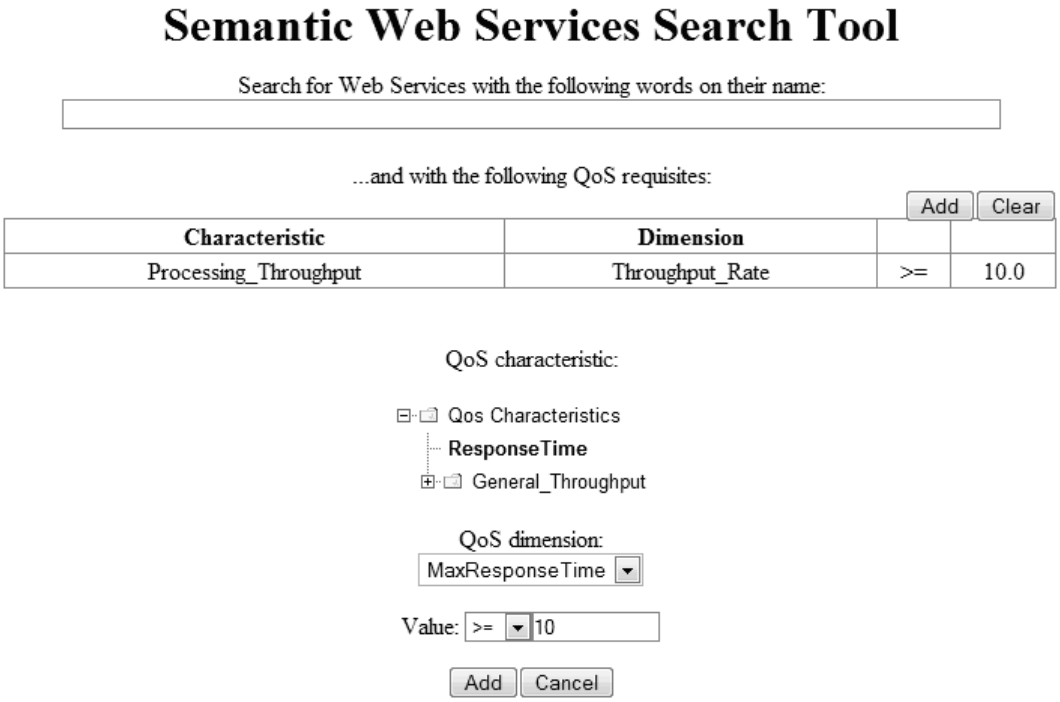

Figura 27: Interface Web para a especificação de Consultas de QoS

Após a execução da consulta, é mostrada em uma nova página uma lista com os Web Services que satisfazem os critérios de busca, conforme é mostrado na Figura 28.

Figura 28: Página Web com os resultados de Consulta baseada em requisitos de QoS

6. ANÁLISE DA ONTOLOGIA QoS-MO

Na continuação, apresentaremos uma análise da expressividade da ontologia QoS-MO, avaliando sua capacidade de expressar todos os elementos necessários para uma completa especificação de QoS e comparando-a com os demais trabalhos relacionados, assim como a avaliação do mecanismo utilizado para descoberta de Web Services. Em seguida, apresentaremos os resultados dos testes de desempenho do mecanismo de busca apresentado no capítulo 5.

6.1. Avaliação da Ontologia

6.1.1. Expressividade

A Ontologia QoS-MO é capaz de expressar todos os elementos e construções que foram identificados em nossos estudos como sendo necessários para uma especificação semântica de QoS completa:

- *Associação de um perfil de QoS com um perfil de Web Service*: esta associação é feita através da associação de restrições de QoS (*QoS Constraints*), que definem o perfil de QoS do Web Service, com o perfil funcional do mesmo. Exemplos de integração com Web Services descritos em OWL-S foram apresentados, porém qualquer outra ontologia de descrição funcional de Web Services ou Componentes de Software poderia ser utilizada integrada à QoS-MO.
- *Definição de um perfil de QoS como um conjunto de características de QoS*: isto é feito através da definição de contextos de QoS, que são uma composição de várias características de QoS.
- *Reuso e extensão de perfis de QoS*: o reuso é alcançado nos modelos de contextos de QoS, restrições de QoS e níveis de QoS através de composições: um contexto de QoS pode ser composto por diversos outros contextos, podendo, assim, aproveitar as definições já existentes; o mesmo ocorre com restrições e níveis. A extensão é conseguida através da possibilidade de criar subclasses de qualquer elemento na ontologia QoS-MO. Por exemplo, a criação de uma sub-

classe de uma classe existente que representa uma característica irá gerar uma nova característica, que estende a existente.

- *Conversão entre unidades de medida*: o modelo de especificação de valores de dimensões permite a escolha da unidade de medida do valor registrado. As unidades de medida são definidas na ontologia, com a possibilidade de especificar fórmulas de conversão entre unidades diferentes.
- *Dimensões compostas*: uma dimensão pode ser definida através da composição de outras dimensões ou outras características.
- *Mapeamento de dimensões*: os mapeamentos são especificados na ontologia através de expressões que indicam as fórmulas para converter os valores de umas dimensões para outras dimensões. Uma ferramenta de suporte pode facilmente calcular os valores não especificados para uma determinada dimensão, desde que exista um mapeamento definido e que tenha os valores das dimensões origem do mapeamento definidos.
- *Flexibilidade na definição de valores*: os valores de dimensões das características podem ter qualquer tipo de dados aceito na linguagem OWL.
- *Dimensão semântica de QoS oferecida e QoS requisitada*: o uso do modelo de restrições de QoS, com as restrições do tipo *QoS Offered* e *QoS Required*, permite ao Serviço especificar quais são as restrições de QoS garantidas por ele e quais são os requisitos que os seus clientes devem atender para que estas garantias sejam válidas, ao mesmo tempo em que permite aos clientes especificarem que requisitos exigem do serviço fornecido e quais são as garantias de qualidade que eles oferecerão ao invocar o serviço.

6.1.2. Mecanismo de descoberta

A ontologia QoS-MO foi construída de forma a permitir que a descoberta de Web Services que atendam a requisitos de QoS especificados pelo cliente possa ser realizada através da simples execução de consultas em linguagem SPARQL. Isto resulta em um mecanismo de fácil implementação e manutenção, já que não há necessidade de criação de um algoritmo complexo.

A necessidade de inferência sobre os dados da ontologia é pequena, limitando-se a estruturas de composição e hierarquia, como explicado na seção 5.1.1. A ontologia foi construída utilizando a linguagem OWL DL, porém no mecanismo de busca foi utiliza-

do apenas um subconjunto das regras de inferência OWL DL, estabelecendo assim um custo de processamento menor do que se fosse utilizado o conjunto completo de regras. Isto contribui para melhorar o desempenho do mecanismo proposto. Adicionalmente, a operação de consulta sobre modelos RDF, executada a partir da consulta SPARQL, também tem desempenho melhor que a inferência.

O estudo mais aprofundado sobre o desempenho do mecanismo será apresentado na seção 6.2.

6.1.3. Comparação com outras ontologias

A Tabela 20 sumariza as informações apresentadas, comparando-as com as características das demais ontologias existentes para especificação de QoS para Web Services Semânticos.

Tabela 20: Comparação entre as ontologias de especificação semântica de QoS

	QoS-MO	DAML-QoS	QoSOnt	OWL-Q
Associação de um perfil de QoS com um perfil de Web Service	Sim	Sim	Sim	Sim
Definição de um perfil de QoS como um conjunto de características de QoS	Sim	Sim	Não	Sim
Reuso e extensão de perfis de QoS	Sim	Sim	Não	Sim
Conversão entre unidades de medida	Sim	Sim	Sim	Sim
Dimensões compostas	Sim	Sim	Não	Sim
Mapeamento de dimensões	Sim	Não	Não	Sim
Flexibilidade na definição de valores	Sim	Não	Sim	Sim
Definição semântica de QoS oferecida e QoS requisitada	Sim	Sim	Não	Sim
Descoberta de QoS	Consulta SPARQL	Inferência DL	UDDI + XML + Lógica	Algoritmo semântico + CSP

A consulta aos dados da ontologia através de consultas SPARQL apresenta uma menor expressividade do que, por exemplo, a utilização de um conjunto completo de

regras de inferência OWL DL. Entretanto, a estrutura proposta para a QoS-MO garante que todas as consultas necessárias para encontrar Componentes de Software que atendam a determinados requisitos possam ser expressadas em SPARQL. De fato, o protótipo do mecanismo de busca implementado executa estas consultas SPARQL para obter os resultados de pesquisa.

Na definição do mecanismo de busca para a QoS-MO, optamos por utilizar este esquema, apesar de sua menor expressividade, apostando no maior desempenho que poderá ser obtido com a realização de simples consultas à ontologia, sem a necessidade de execução de um mecanismo de inferência para encontrar os Componentes que atendam a requisitos específicos de QoS. Além disso, esta abordagem também tira proveito da adoção de um padrão W3C, a linguagem SPARQL, ao invés da criação de novos mecanismos, o que deverá facilitar a adoção da metodologia proposta.

6.2. Desempenho do mecanismo de busca

Alguns testes foram realizados para avaliar o desempenho do mecanismo de busca, de modo a permitir uma análise da eficiência da estratégia empregada na sua construção. Para a realização dos testes, foi gerada artificialmente uma Ontologia com descrições de QoS de Web Services, variando de 500 a 10.000 Web Services.

Os testes foram efetuados em um computador com processador Intel Core 2 Duo 2,4 GHz e 2 GB de memória RAM, rodando o sistema operacional Fedora 7 e o JRE 6 da SUN. Todos os testes foram efetuados através de consultas locais, eliminando a possibilidade de interferência de tempos de transmissão de rede e levando em consideração apenas o tempo de processamento do servidor para realizar a consulta e montar a lista de resultados.

6.2.1. Testes de tempo de resposta e uso de memória

Os primeiros testes realizados avaliaram o tempo de carregamento da ontologia de domínio. Os testes mostraram que a carga utilizando o conjunto de inferência Micro OWL da API Jena é efetuada em um tempo aceitável. Quando uma ontologia com 10.000 componentes é utilizada, o tempo de carga é de pouco mais de 10 segundos. Como esta operação só é realizada na carga do sistema, este tempo é considerado baixo.

Um segundo teste foi efetuado com o intuito de avaliar o desempenho da busca. Foi possível verificar nesse teste que, ao aumentarmos o número de componentes regis-

trados em uma determinada proporção, o tempo de busca cresce em uma proporção menor. Conforme mostrado na Tabela 21, a busca em uma ontologia com 1.000 componentes leva aproximadamente 1,6 segundo para ser realizada; se aumentarmos dez vezes o número de componentes registrados, o tempo de busca passa a ser de aproximadamente 5 segundos, ou seja, menos de quatro vezes mais.

Tabela 21: Resultado dos testes de desempenho e alocação de memória

Quantidade de Web Services	Tempo de carga (s)	Tempo de pesquisa (s)	Uso de Memória (MB)
500	1,0	0,8	35
750	1,3	1,2	39
1.000	1,7	1,6	45
2.500	2,9	1,8	45
5.000	5,7	3,7	65
7.500	8,0	3,9	96
10.000	10,8	4,9	126

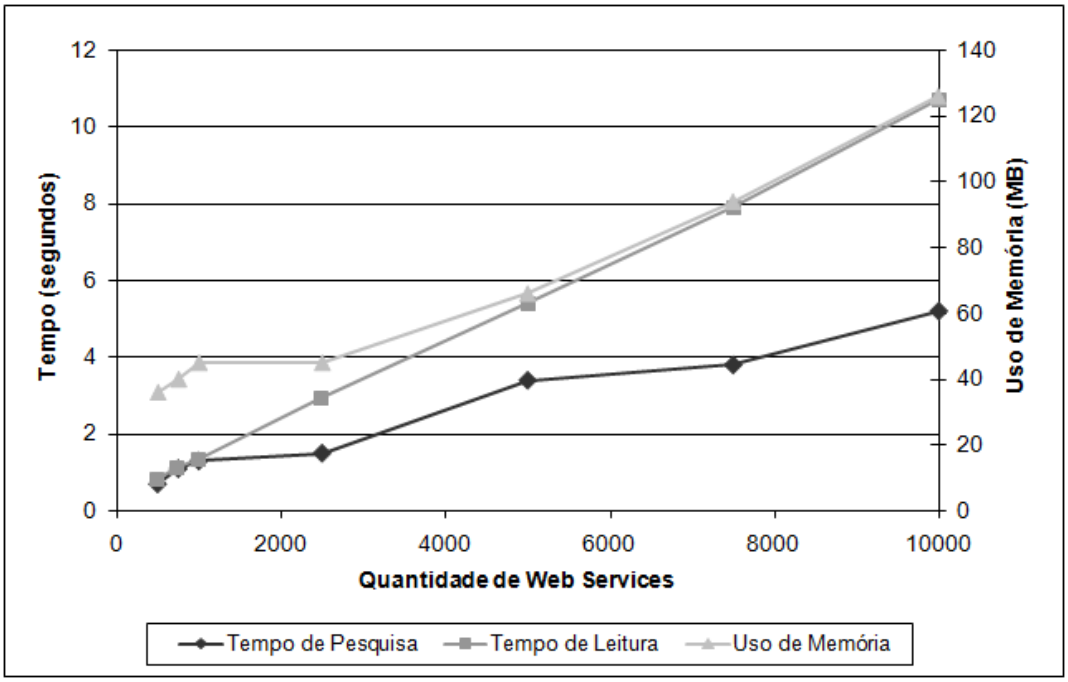

Figura 29: Resultado dos testes de desempenho e alocação de memória

Os resultados dos testes, sumarizados na Figura 29, também mostram a quantidade de memória necessária para armazenar a ontologia e os resultados da busca. Também

se verificou que o uso de memória cresce a uma proporção menor que o número de componentes: com 1.000 componentes, o uso foi de 45 MB; aumentando dez vezes o número de componentes, o uso de memória foi para 126 MB, ou seja, pouco menos de três vezes mais.

6.2.2. Testes de execução concorrente

Testes adicionais foram realizados para verificar a eficiência do mecanismo de busca com consultas concorrentes. Para isto, além de variar a quantidade de Web Services na ontologia, foram feitos testes que variaram de 50 a 1.000 *threads* simultâneas de consulta.

Os resultados, apresentados na Tabela 22 e sumarizados na Figura 30, mostram que a eficiência da busca diminui conforme vai aumentando o tamanho da Ontologia ou a quantidade de consultas simultâneas, porém em proporção menor do que o aumento registrado da carga.

Embora nossa máquina de teste não fosse um servidor preparado para suportar um grande número de requisições simultâneas, ele foi capaz de manter um bom nível de consultas por segundo, chegando a um máximo de 77 consultas por segundo para realizar 1.000 consultas a uma Ontologia com 2.500 componentes e a um mínimo de 7 consultas por segundo para realizar 1.000 consultas a uma Ontologia com 10.000 componentes.

Tabela 22: Resultado dos testes de desempenho com consultas concorrentes

Quantidade de Web Services	**Quantidade de Threads de consulta / Tempo de pesquisa (s)**					
	50	**100**	**250**	**500**	**750**	**1.000**
500	2,1	2,5	2,5	3,8	4,2	5,2
750	4,7	5,0	8,7	14,0	18,9	24,3
1.000	3,2	4,3	7,5	10,7	14,8	19,0
2.500	2,6	3,7	4,	6,9	9,8	11,8
5.000	7,7	11,0	20,6	38,8	54,6	68,8
7.500	4,3	6,0	9,9	15,7	22,7	28,4
10.000	12,3	19,8	37,9	70,7	102,5	141,6

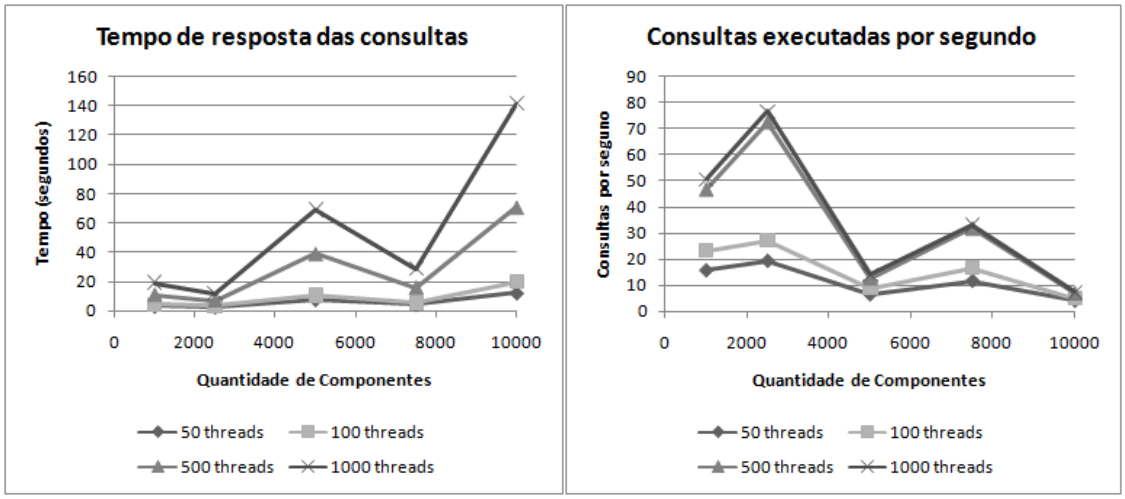

Figura 30: Resultado dos testes de desempenho com consultas concorrentes

O programa utilizado para realizar o teste era simples e criou cada uma das 1.000 threads de consulta uma a uma. A implementação de algum mecanismo de *pooling* de threads, comumente utilizado em servidores preparados para suportar altas taxas de requisições simultâneas, provavelmente melhoraria ainda mais o desempenho obtido.

Uma observação efetuada é que, à medida que se aumenta a quantidade de Componentes descritos na Ontologia, o tempo necessário para efetuar as buscar não é sempre crescente: por vezes o tempo aumenta, outras vezes o tempo diminui ao incluir mais componentes e depois volta a aumentar. Por exemplo, os resultados para se efetuar consultas a 7.500 componentes foram bem melhores do que para 5.000 componentes.

Este comportamento foi confirmado executando o teste várias vezes, em computadores diferentes e com variações da consulta SPARQL executada, sempre com o mesmo resultado. Provavelmente, isto deve ser causado por algum detalhe específico da implementação do mecanismo de consulta SPARQL da API ARQ.

Um trabalho a ser realizado no futuro é analisar melhor os detalhes deste comportamento do mecanismo de consulta SPARQL da API ARQ e avaliar a possibilidade de aperfeiçoá-lo para que seja possível obter sempre resultados nos níveis dos melhores tempos de resposta apresentados pelo mecanismo.

7. CONCLUSÕES

A seguir, apresentaremos os resultados obtidos com a realização do presente trabalho e, em seguida, as perspectivas para a continuidade do mesmo.

7.1. Resultados alcançados

Este trabalho desenvolveu a ontologia QoS-MO para a descrição detalhada de QoS de Web Services Semânticos ou Componentes de Software em geral, assim como um mecanismo de descoberta de Web Services Semânticos baseado em QoS e implementado através da utilização da linguagem de consulta SPARQL.

A construção da ontologia QoS-MO foi baseada em vários modelos existentes, especialmente no *Framework* de QoS da OMG. A QoS-MO pode estender qualquer ontologia de descrição funcional de Componentes de Software ou Web Services, acrescentando a capacidade de especificação de Características e Restrições de QoS. Foi desenvolvido, ao longo do trabalho, o suporte para extensão da OWL-S, escolhida como padrão de ontologia para a descrição funcional de Web Services devido ao fato de já ser um padrão reconhecido mundialmente, inclusive pela W3C, com grande possibilidade de aplicação prática.

As especificações criadas com QoS-MO podem ser convertidas para um *Profile* UML de QoS ou importadas de um *Profile* UML, permitindo a integração de modelos de QoS na fase de projeto do ciclo de vida de desenvolvimento de um Componente de Software.

A ontologia QoS-MO conta com padrões para especificação de Características e Contextos de QoS, Restrições de QoS e Níveis de QoS. Ao longo do trabalho apresentamos os detalhes de cada um destes modelos e exemplos de sua utilização.

Os modelos propostos, mesmo se mantendo simples, são capazes de especificar todos os detalhes de características de QoS complexas identificados em nosso estudo, como composição e mapeamento de dimensões, extensão e reutilização de perfis de QoS, e diferentes níveis de QoS com as características de QoS garantidas e os requisitos para operação em cada nível. Comparando com as demais propostas de ontologia para

descrição de QoS existentes, a QoS-MO foi uma das duas únicas capazes de alcançar este nível de expressividade.

Também desenvolvemos um mecanismo semântico de busca com base em requisitos de qualidade de serviço, que pode ser utilizado para a seleção com base em atributos de QoS de Componentes de Software ou Web Services Semânticos. O mecanismo foi construído sobre uma API de busca que permite a identificação de características de QoS em uma ontologia e executa buscas para localização de Componentes ou Web Services que satisfaçam determinados requisitos de QoS. A ontologia QoS-MO é empregada para descrever as características e restrições de QoS. Toda a construção da ontologia teve o cuidado de criar estruturas que permitissem esta abordagem de descoberta.

A abordagem para a descoberta de Serviços adotada neste trabalho é simples e de fácil implementação e manutenção. As buscas são executadas montando uma consulta SPARQL com base nos requisitos de QoS especificados pelo cliente e executando-a sobre a ontologia que descreve os Componentes ou Web Services disponíveis. A ontologia de descrição de QoS passa previamente por um processo de inferência para identificar as estruturas de especialização e composição de especificações.

Uma interface Web de consulta foi desenvolvida para demonstrar a viabilidade da abordagem proposta e testes foram realizados com o intuito de avaliar o desempenho do mecanismo de busca. Os resultados dos testes mostram que a abordagem proposta é viável e foi possível obter tempos de resposta aceitáveis mesmo para ontologias com milhares de Web Services e executando centenas de consultas simultaneamente.

Enquanto outras propostas existentes demandam a construção de algoritmos complexos ou a utilização de recursos externos para identificar os Web Services que atendem aos requisitos de QoS especificados, a abordagem proposta neste trabalho depende apenas da capacidade de realização de inferência simples e consultas SPARQL, que são recursos que têm se tornado disponíveis com cada vez mais facilidade. Apesar de esta abordagem apresentar menor expressividade do que, por exemplo, a utilização de um conjunto completo de regras de inferência OWL DL, a estrutura proposta para a QoS-MO garante que todas as consultas necessárias possam ser expressadas em SPARQL, além do benefício de estarmos utilizando apenas padrões já estabelecidos mundialmente para a criação de nosso mecanismo.

Durante o desenvolvimento do trabalho, três artigos científicos foram publicados para apresentar os resultados alcançados, sendo dois em eventos internacionais (TONDELLO & SIQUEIRA, 2008a e 2008c) e um em evento nacional (TONDELLO & SIQUEIRA, 2008b).

7.2. Perspectivas futuras

Futuramente, a ferramenta de busca deverá ser submetida a mais testes em ambientes de maior escala, nos quais seja necessário atender a uma grande quantidade de requisições em paralelo, com o intuito de melhor avaliar e aperfeiçoar o seu desempenho em situações de sobrecarga. Além disso, os resultados dos testes de desempenho com consultas concorrentes, apresentados no capítulo 6, deverão ser melhores analisados, visando encontrar explicações para as flutuações observadas e desenvolver os meios para a obtenção sempre do melhor desempenho possível.

Também seria interessante a realização de estudos de desempenho com a Ontologia de Componentes mantida em um Banco de Dados relacional ao invés de mantida na memória, recurso suportado pela API Jena – utilizada para construção do mecanismo de busca – para permitir a escalabilidade da solução proposta.

A comparação entre estudos de desempenho desta e das demais propostas de trabalhos relacionados não foi possível, por não se encontrar na literatura dados objetivos sobre o desempenho de cada uma das demais propostas. A realização de estudos deste tipo será importante para que se possam avaliar as diferenças na eficiência alcançada por cada uma das idéias existentes na área.

O mecanismo de busca de Web Services que foi desenvolvido só é capaz de identificar se os mesmos atendem totalmente ou parcialmente aos requisitos especificados, não realizando nenhuma classificação adicional dos resultados de pesquisa. A criação de um mecanismo de classificação dos resultados encontrados, identificando quais Serviços atendem melhor ou com mais folga aos requisitos, seria um trabalho importante, já que outras abordagens existentes são capazes de realizar isto.

Este trabalho teve como foco a especificação e seleção de Componentes e Serviços com base em seus parâmetros de qualidade de serviço. Entretanto, a consideração dos parâmetros de QoS só faz sentido quando sabe-se que o Componente atende aos requisitos funcionais que se necessita. Para que a abordagem desenvolvida possa ser

utilizada na prática, ela deverá futuramente ser integrada a algum mecanismo de busca semântica de Componentes ou Web Services baseado em suas características funcionais.

A ontologia QoS-MO pode ser utilizada em conjunto com qualquer ontologia de descrição funcional de Componentes de Software ou Web Services, porém somente a integração com OWL-S foi desenvolvida neste trabalho. Futuramente, poderão ser realizados estudos visando à integração com outras ontologias padrão de especificação existentes na literatura como, por exemplo, a WSMO.

Para viabilizar a utilização prática da abordagem proposta, será necessário desenvolver melhor o mecanismo de busca, que no momento é somente um protótipo e não conta com a robustez e a flexibilidade requeridas para ser utilizado em grande escala. Também será muito importante o desenvolvimento de um mecanismo e de uma interface com o usuário para a publicação de componentes em uma ontologia baseada em QoS-MO, possibilitando que a publicação de componentes seja realizada sem que seja necessário conhecer e acessar diretamente a ontologia. O mecanismo de publicação de componentes também deverá realizar todas as tarefas que são necessárias em tempo de publicação, como o processamento do mapeamento de dimensões, conforme apresentado no capítulo 4.

Finalmente, a utilização da QoS-MO para a especificação de Web Services em ambientes reais permitirá identificar se existem aspectos ou exceções não contemplados na definição da ontologia e possibilitará futuros estudos para a inclusão de novas construções ou adaptação das existentes para melhorar a expressividade da ontologia.

BIBLIOGRAFIA

BAADER, F.; CALVANESE, D.; MCGUINNESS, D.; et al., editores. **The Description Logic Handbook:** Theory, Implementation, and Applications. EUA: Cambridge University Press, 2003.

BERNERS-LEE, Tim; HENDLER, James; LASSILA, Ora. The Semantic Web. **Scientific American**, Maio 2001.

BERNERS-LEE, Tim. **Artificial Intelligence and the Semantic Web: AAAI2006 Keynote**. Julho 2006. http://www.w3.org/2006/Talks/0718-aaai-tbl/Overview.html

BROWN, N.; KINDEL, C. **Distributed component object model protocol, DCOM/1.0**. Redmond, EUA: Microsoft Corporation, 1996.

BRUSSEE, Rogier; POKRAEV, Stanislav. Reasoning on the Semantic Web. Em: CARDOSO, Jorge; editor. **Semantic Web Services**: Theory, Tools, and Applications. EUA: Information Science Reference, 2007, pág. 110-133. ISBN 978-1-59904-047-9.

CIMPIAN, E.; MORAN, M.; OREN, E.; et al. **Overview and Scope of WSMX**. Technical Report, WSMX Working Draft. Fevereiro 2005. http://www.wsmo.org/TR/d13/d13.0/v0.2/

CONNOLY, Dan; HARMELEN, Frank van; HORROCKS, Ian; et al. **DAML+OIL Reference Description**. Março 2001. http://www.w3.org/TR/daml+oil-reference

CURBERA, Francisco; NAGY, William; WEERAWARANA, Sanjiva. Web Services: Why and How. **Workshop on Object-Oriented Web Services** (OOPSLA 2001). Florida, EUA, 2001.

DAML PROGRAM. **About the DAML Language**. Fevereiro 2001. http://www.daml.org/about.html

DAVIES, John; STUDER, Rudi; WARREN, Paul; editores. **Semantic Web Technologies:** Trends and Research in Ontology-based Systems. Inglaterra: John Wiley & Sons, 2006, pág. 1-8. ISBN 0-470-02596-4.

DE BRUIJN, Jos; editor. **The Web Service Modeling Language WSML**. WSMO Final Draft v0.2. 2005. http://www.wsmo.org/TR/d16/d16.1/v0.2/

DOBSON, Glen; LOCK, Russell; SOMMERVILLE, Ian. QoSOnt: a QoS Ontology for Service-Centric Systems. **31st Euromicro Conference on Software Engineering and Advanced Applications** (Euromicro SEAA '05). Porto, Portugal, 2005.

DWONING, T. **Java RMI**. Boston, EUA: IDG Books Worldwide, 1998.

ESSI WSMO working group. **Web Service Modeling Ontology (WSMO) Final Draft 21**. Outubro 2006. http://www.wsmo.org/TR/d2/v1.3/

FENSEL, Dieter; HARMELEN, Frank van; HORROCKS, Ian; et al. OIL: An Ontology Infrastructure for the Semantic Web. **IEEE Intelligent Systems**. Março/Abril 2001, Vol. 16, Nº 2.

FENSEL, Dieter; LAUSEN, Holger; POLLERES, Axel; et al. **Enabling Semantic Web Services**: The Web Service Modeling Ontology. Alemanha: Springer, 2007. ISBN 3-540-34519-1.

FRØLUND, S.; KOISTINEN, J. **QML: A Language for Quality of Service Specification**. 1998. http://www.hpl.hp.com/techreports/98/HPL-98-10.html

GRUBER, Tom R. A translation approach to portable ontologies. **Knowledge Acquisition**, 1993, Vol. 5, Nº 2, pág. 199-220. http://tomgruber.org/writing/ontolingua-kaj-1993.htm

HITZLER, Pascal; ANGELE, Jürgen; MOTIK, Boris; STUDER, Rudi. Bridging the Paradigm Gap with Rules for OWL. **Proceedings of the W3C Workshop on Rule Languages for Interoperability**. http://www.w3.org/2004/12/rules-ws/

HORROCKS, Ian; PATEL-SCHNEIDER, Peter F.; BOLEY, Harold; et al. **SWRL: A Semantic Web Rule Language Combining OWL and RuleML**. Maio 2004. http://www.w3.org/Submission/SWRL/

ISO (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION). **Information Technology – Quality of Service – Framework**. ISO/IEC 13236, 1998.

JENA – A Semantic Web Framework for Java. 2008. http://jena.sourceforge.net/

KELLER, Alexander; LUDWIG, Heiko. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. **Journal of Network and Systems Management**, 2003, Vol. 11, Nº 1, pág. 57-81.

KRITIKOS, Kyriakos; PLEXOUSAKIS, Dimitris. Semantic QoS Metric Matching. **4th European Conference on Web Services** (ECOWS '06), Dezembro 2006, pág. 265-274.

LUDWIG, H.; KELLER, A.; DAN, A.; et al. **Web Service Level Agreement (WSLA) Language Specification**. 2003. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

MCGUINNESS, D.L.; FIKES, R.; HENDLER J.; STEIN, L.A., DAML+OIL: An Ontology Language for the Semantic Web. **IEEE Intelligent Systems**, 2002.

MCILRAITH, S.; SON, T.C.; ZENG, H. Semantic Web Services. **IEEE Intelligent Systems**: Special Issue on the Semantic Web. 2001, Vol. 16, Nº2, pág. 46-53.

MENASCÉ, Daniel A. QoS issues in Web services. **IEEE Internet Computing**, 2002, Vol. 6, Nº 6, pág. 72-75.

NAHRSTEDT, Klara; STEINMETZ, Ralf. Resource Management in Multimedia Networked Systems. **IEEE Computer**, Maio 1995, pág. 52-64.

OASIS. **UDDI Version 3.0.2**. 2004. http://www.uddi.org/pubs/uddi_v3.htm

OMG (OBJECT MANAGEMENT GROUP). **CORBA: The Common Object Request: Architecture and Specification, Release 2.0**. Julho 1995. http://www.omg.org/cgi-bin/apps/doc?formal/99-10-07.pdf

OMG (OBJECT MANAGEMENT GROUP). **UML Profile for Modeling QoS and FT Characteristics and Mechanisms Specification, v1.0**. Maio 2006.

OWL-S COALITION. **OWL-S 1.1 Release**. Novembro 2004. http://www.daml.org/services/owl-s/1.1/

PENNINGTON, Cary; CARDOSO, Jorge; MILLER, John A.; et al. Introduction to Web Services. Em: CARDOSO, Jorge; editor. **Semantic Web Services**: Theory, Tools, and Applications. EUA: Information Science Reference, 2007, pág. 134-154. ISBN 978-1-59904-047-9.

ROMAN, Dumitru; DE BRUIJN, Jos; MOCAN, Adrian; et al. Semantic Web Services – Approaches and Perspectives. Em: DAVIES, John; STUDER, Rudi; WARREN, Paul; editores. **Semantic Web Technologies:** Trends and Research in Ontology-based Systems. Inglaterra: John Wiley & Sons, 2006, pág. 191-236. ISBN 0-470-02596-4.

Semantic Web Services Challenge. 2006. http://www.sws-challenge.org/

SIQUEIRA, Frank. Especificação de Requisitos de Qualidade de Serviço em Sistemas Abertos: A Linguagem QSL. **20º Simpósio Brasileiro de Redes de Computadores** (SBRC‘2002). Búzios - RJ, Brasil, 2002.

STAAB, S.; STUDER, R.; editores. International Handbooks on Information Systems: **Handbook on Ontologies**. Springer, 2004. ISBN 3-540-40834-7.

TONDELLO, Gustavo F.; SIQUEIRA, Frank. The QoS-MO Ontology for Semantic QoS Modeling. **Proceedings of the 23rd Annual ACM Symposium on Applied Computing** (ACM SAC 2008). Fortaleza - CE, Brasil, Março 2008 (a), pág. 2336-2340.

TONDELLO, Gustavo F.; SIQUEIRA, Frank. Um Mecanismo Semântico de Busca de Componentes de Software Baseado em Qualidade de Serviço. **II Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software** (SBCARS 2008). Porto Alegre - RS, Brasil, Agosto 2008 (b).

TONDELLO, Gustavo F.; SIQUEIRA, Frank. A QoS-based Search Engine for Semantic Web Services. **XXXIV Conferencia Latinoamericana de Informática** (CLEI 2008). Santa Fe, Argentina, Setembro 2008 (c).

VAN HENTENRYCK, P.; SARASWAT, V. Strategic directions in constraint programming. **ACM Computing Surveys**, 1996, Vol. 28, Nº 4, pág. 701–726.

VERMA, Dinesh. Service Level Agreements on IP Networks. **Proceedings of the IEEE**, Setembro 2004, Vol. 92, Nº 9, pág. 1382-1388.

VOGEL, Andreas; KERHERVÉ, Brigitte; VON BOCHMANN, Gregor; GECSEI, Jan. Distributed Multimedia and QoS: A Survey. **IEEE Multimedia**. 1995, Vol. 2, Nº 2.

VU, Le-Hung; HAUSWIRTH, Manfred; PORTO, Fabio; ABERER, Karl. A Search Engine for QoS-enabled Discovery of Semantic Web Services. **Special Issue of the International Journal on Business Process Integration and Management** (IJBPIM), 2006, Vol. 1, Nº 4, pág. 244–255.

W3C (WORLD WIDE WEB CONSORTIUM). **Extensible Markup Language (XML)**. 2003 (a). http://www.w3.org/XML/

W3C (WORLD WIDE WEB CONSORTIUM). **SOAP Version 1.2**. 2003 (b). http://www.w3.org/TR/soap12

W3C (WORLD WIDE WEB CONSORTIUM). **QoS for Web Services: Requirements and Possible Approaches**. W3C Working Group Note. 2003 (c). http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/

W3C (WORLD WIDE WEB CONSORTIUM). **Resource Description Framework (RDF)**. 2004 (a). http://www.w3.org/RDF/

W3C (WORLD WIDE WEB CONSORTIUM). **Web Ontology Language (OWL)**. 2004 (b). http://www.w3.org/2004/OWL/

W3C (WORLD WIDE WEB CONSORTIUM). **Web Services Glossary**. 2004 (c). http://www.w3.org/TR/ws-gloss/

W3C (WORLD WIDE WEB CONSORTIUM). **RIF RDF and OWL Compatibility**. 2005. http://www.w3.org/2005/rules/wiki/SWC

W3C (WORLD WIDE WEB CONSORTIUM). **SPARQL Query Language for RDF**. Candidate Recommendation. 2007 (a). http://www.w3.org/TR/rdf-sparql-query/

W3C (WORLD WIDE WEB CONSORTIUM). **Web Services Description Language (WSDL) Version 2.0**. 2007 (b). http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/

WEIBEL, S.; KUNZE, J.; LAGOZE, C.; WOLF, M. **Dublin core metadata for resource discovery**. RFC 2413, IETF. Setembro 1998.

WSMO Working Group. **Web Service Modeling Ontology**. 2008. http://www.wsmo.org/

ZHOU, Chen; CHIA, Liang-Tien; LEE, Bu-Sung. DAML-QoS ontology for Web services. **IEEE International Conference on Web Services** (ICWS‘04). San Diego, California, EUA, 2004, pág. 472-479.

ZINKY, J.; BAKKEN, D; SCHANTZ, R. Architectural Support for Quality of Service for CORBA Objects. **Theory and Practice of Object Systems**. 1997, Vol. 3, Nº 1.